Introdução à Estatística para Ciência de Dados

Da exploração dos dados à experimentação contínua com exemplos de código em Python e R

Casa do Código | alura

TATIANA ESCOVEDO
THIAGO MARQUES
MARCOS KALINOWSKI

Sumário

- ISBN
- Sobre o livro
- Sobre os autores
- 1. Introdução
- 2. Estatística Descritiva: Conceitos básicos, tipos de variáveis e gráficos
- 3. Estatística Descritiva: Medidas de tendência central e de dispersão
- 4. Cálculo das Probabilidades: Conceitos e fundamentos
- 5. Cálculo das Probabilidades: Distribuições de probabilidade discretas e contínuas
- 6. Inferência Estatística: Noções de amostragem e reamostragem
- 7. Inferência Estatística: Modelagem Estatística - Introdução aos algoritmos de Machine Learning e comparação de modelos
- 8. Experimentação Contínua

ISBN

Impresso: 978-85-5519-380-4

Digital: 978-85-5519-379-8

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Escovedo, Tatiana
Introdução à estatística para ciência de dados : da exploração dos dados à experimentação contínua com exemplos de código em Python e R / Tatiana Escovedo, Marcos Kalinowski, Thiago Marques. -- 1. ed. -- São Paulo : AOVS Sistemas de Informática, 2024.

Bibliografia.
ISBN 978-85-5519-380-4

1. Ciência da Computação 2. Dados - Estruturas (Ciência da computação) 3. Estatística 4. Estatística - Estudo e ensino 5. Python (Linguagem de programação para computadores) I. Kalinowski, Marcos. II. Marques, Thiago. III. Título.

24-211500 CDD-519.5

Índices para catálogo sistemático:

1. Estatística 519.5

Aline Graziele Benitez - Bibliotecária - CRB-1/3129

A arte da capa deste livro foi desenvolvida pela NoClaf.

Caso você deseje submeter alguma errata ou sugestão, acesse http://erratas.casadocodigo.com.br.

Sobre o livro

Por que escrevemos este livro?

Quando eu, Tatiana, iniciei meus estudos em Ciência de Dados, há muitos anos, sentia que a maioria dos livros existentes na literatura era muito complexa, recheada de demonstrações matemáticas e detalhes técnicos muitas vezes assustadores para um iniciante na área, especialmente para aqueles cuja base matemática não era tão profunda. Cada pessoa tem um estilo preferido de aprendizagem e, apesar de muitos preferirem conteúdos detalhados e profundos, o estilo que sempre funcionou para mim foi o que eu chamo de "aprendizado cebola", pois é feito em camadas: gosto de ter primeiro uma visão geral de todos os assuntos e, depois, ir me aprofundando em determinados tópicos de acordo com a minha necessidade. Meus alunos adaptaram o conceito de "aprendizado cebola" para: quanto mais você aprende e se aprofunda, mais lágrimas você derrama. Acredito que estas lágrimas se refiram às lágrimas de alegria de estudar Ciência de Dados!

Pensando também que a maioria dos livros técnicos é escrita em inglês e muitos estudantes têm dificuldades de compreensão do idioma (ou mesmo preferem ler em português) surgiu a ideia de escrever um livro introdutório de Ciência de Dados em português, que pudesse guiar os aprendizes iniciantes nesta área fascinante, e estimulá-los a se desenvolverem em tópicos mais avançados. Esse livro foi lançado em 2020 pela Editora Casa do Código e você pode conferir em https://www.casadocodigo.com.br/products/livro-data-science.

Nesta linha, tive a ideia de convidar alguns profissionais incríveis para juntos escrevermos um livro utilizando a mesma abordagem, mas agora focado em conceitos básicos de Estatística necessários para uma pessoa cientista de dados. Além dos conceitos, apresentamos exemplos práticos nas linguagens R e Python.

Esperamos que a leitura seja agradável e que este livro possa ser útil na sua caminhada! Bons estudos.

Para quem é este livro?

Este livro destina-se a profissionais e estudantes da área de Ciência de Dados, que estejam interessados em aprender conceitos básicos de Estatística que serão úteis para um cientista de dados para trabalhar, por exemplo, com análise exploratória de dados e Machine Learning. Assim, o livro não se destina a trazer um conteúdo completo sobre Ciência de Dados. Para um estudo mais completo (porém, não exaustivo, devido à complexidade da área), recomendamos as seguintes referências:

- *Introdução a Data Science: Algoritmos de Machine Learning e métodos de análise*, de Tatiana Escovedo e Adriano Koshiyama
- *Engenharia de Software para Ciência de Dados: Um guia de boas práticas com ênfase na construção de sistemas de Machine Learning em Python*, de Marcos Kalinowski, Tatiana Escovedo, Hugo Villamizar e Hélio Lopes
- *Introduction to Data Science: A Python Approach to Concepts, Techniques and Applications*, de Laura Igual e Santi Seguí
- *An Introduction to Statistical Learning*, de Gareth James, Daniela Witten, Trevor Hastie e Robert Tibshirani

Além disso, você pode encontrar uma lista (em constante construção) de indicações de materiais de estudo na área de Ciência de Dados (e áreas relacionadas) no link https://www.linkedin.com/pulse/material-de-estudo-ci%C3%AAncia-dados-tatiana-escovedo-phd/.

Este livro também assume que você conhece os conceitos básicos de lógica de programação e entende o básico de programação em Python e R e de suas principais bibliotecas, uma vez que os exemplos serão apresentados nestas linguagens. Para uma rápida introdução a estas linguagens, recomendamos os links https://learnxinyminutes.com/docs/python/ e

https://learnxinyminutes.com/docs/r/. Para aprofundar-se e praticar, a *W3Schools* tem excelentes tutoriais de Python e R online, disponíveis, respectivamente, em https://www.w3schools.com/python/ e https://www.w3schools.com/r/. Se você quiser aprofundar seus estudos nestas linguagens, há na literatura diversos livros disponíveis recomendados, tais como *Python Fluente: Programação Clara, Concisa e Eficaz*, *Python para Análise de Dados: Tratamento de dados com Pandas, NumPy e IPython* e *R para Data Science: Importe, arrume, transforme, visualize e modele dados*. Além disso, a **Editora Casa do Código** oferece diversos livros que abordam Python, disponíveis em https://www.casadocodigo.com.br/collections/programacao-python. Se você julgar que precisa de um reforço em lógica de programação, veja os livros disponíveis em https://www.casadocodigo.com.br/collections/programacao-logica. Também recomendamos os excelentes livros da série **Use a Cabeça**: *Use a Cabeça! Aprenda a Programar* e *Use a Cabeça! Programação*.

O principal objetivo deste livro é apresentar alguns dos principais conceitos da Estatística aplicados a Ciência de Dados de forma introdutória, a fim de capacitar os cientistas de dados nesta área de conhecimento. Portanto, é importante ressaltar que este livro não tem o objetivo de ser uma literatura exaustiva ou completa sobre Estatística geral, nem um livro texto para cursos de Estatística, uma vez que já existem diversos livros consolidados e recomendados sobre o assunto. Para aqueles que tiverem interesse em outros tópicos da Estatística não abordados neste livro, recomendamos as seguintes referências:

- *Introdução à Estatística*, de Mario Triola
- *Estatística Básica*, de Pedro Morettin e Wilton Bussab
- *Estatística Aplicada e Probabilidade para Engenheiros*, de Douglas Montgomery e George Runger

Como este livro está organizado?

Este livro aborda os principais conceitos de Estatística Descritiva, Probabilidade e Inferência Estatística relacionados a Ciência de Dados, e está organizado em oito capítulos, a saber:

- Capítulo 1: faz uma introdução ao tema, abordando a motivação de se estudar Estatística para Ciência de Dados e alguns conceitos básicos fundamentais;
- Capítulo 2: focado na Estatística Descritiva, apresenta conceitos básicos, tipos de variáveis e gráficos;
- Capítulo 3: ainda focado na Estatística Descritiva, aborda medidas de tendência central e de dispersão;
- Capítulo 4: focado em Probabilidade, apresenta conceitos e fundamentos;
- Capítulo 5: ainda focado no cálculo das Probabilidades, aborda distribuições de probabilidade discretas e contínuas;
- Capítulo 6: focado na Inferência Estatística, apresenta noções de amostragem e reamostragem;
- Capítulo 7: ainda focado na Inferência Estatística, em especial na Modelagem Estatística, introduz conceitos sobre algoritmos de Machine Learning e comparação de modelos;
- Capítulo 8: focado em Experimentação Contínua, aborda conceitos de teste de hipótese para subsidiar decisões referentes à implantação de modelos de Machine Learning.

Recursos

Os exemplos deste livro encontram-se nas linguagens Python e/ou R, as linguagens atualmente mais utilizadas pelos cientistas de dados, e têm o objetivo de ilustrar, de forma prática, os pontos teóricos abordados em cada capítulo. Recomendamos que você leia este livro sequencialmente e execute cada exemplo à medida que fizer a leitura dos capítulos. É possível utilizar diversos ambientes e IDEs para codificar nestas linguagens, tais como o PyCharm para Python e o RStudio para o R.

Para fins de simplificação, especialmente para o(a) cientista de dados iniciante, recomendamos a utilização do **Colab**

(https://colab.research.google.com/), uma ferramenta da Google que permite escrever e executar código em Python usando o próprio navegador, sem necessidade de realizar nenhuma instalação adicional. O Colab também oferece acesso gratuito a GPUs e permite compartilhar facilmente os seus códigos com terceiros. O Colab possibilita mesclar blocos de texto e código, gerando um *notebook* com código documentado. Também é possível codificar em R usando o Colab, bastando executar o comando `%load_ext rpy2.ipython` e, em seguida, iniciar os blocos de código em R com a instrução `%%R` .

> Os exemplos de código deste livro podem ser encontrados no seu repositório do GitHub, https://github.com/tatianaesc/estatisticadatascience.

Referências bibliográficas (livros indicados)

Escovedo, T. e Koshiyama, A. *Introdução a Data Science:* Algoritmos de Machine Learning e métodos de análise. São Paulo: Casa do Código, 2020.

Igual, L. e Seguí, S. *Introduction to Data Science:* A Python Approach to Concepts, Techniques and Applications. Springer, 2017.

James, G., Witten, D., Hastie, T. e Tibshirani, R. *An introduction to Statistical Learning*. Springer, 2013.

Kalinowski, M., Escovedo, T., Villamizar, H. e Lopes, H. *Engenharia de Software para Ciência de Dados:* Um guia de boas práticas com ênfase na construção

de sistemas de Machine Learning em Python. São Paulo: Casa do Código, 2023.

Montgomery, D. e Runger, G. *Estatística Aplicada e Probabilidade para Engenheiros*. LTC, 7ª edição, 2021.

Morettin, P. e Bussab W. *Estatística Básica*. Saraiva Uni, 9ª edição, 2017.

Triola, Mario F. *Introdução à estatística*. LTC, 12ª edição, 2017.

Sobre os autores

Tatiana Escovedo

Tatiana Escovedo é Gerente Geral de Transformação Digital da Petrobras. Além disso, atua como professora do Departamento de Informática da PUC-Rio, onde coordena a Pós-Graduação lato sensu de Ciência de Dados e Analytics e colabora com pesquisas nas áreas de Ciência de Dados e Engenharia de Software. É doutora em Engenharia Elétrica, na área de Métodos de Apoio à Decisão, mestre em Informática na área de Engenharia de Software e bacharel em Sistemas de Informação, todos pela PUC-Rio. É autora de diversos livros e artigos na sua área de atuação.

Thiago Marques

Thiago Marques é Analista Censitário em Métodos Quantitativos no IBGE e CEO da Comunidade de Estatística do prof. Thiago Marques. É bacharel em Estatística pela ENCE/IBGE (Escola Nacional de Ciências Estatísticas). Nos últimos anos, atuou como coordenador e professor em MBAs e Pós-Graduações lato sensu na área de dados. Possui experiência como consultor em estatística e Analytics em diversas empresas. É criador do maior canal do YouTube de Estatística e Ciência de Dados da América Latina, o *EstaTiDados*.

Marcos Kalinowski

Marcos Kalinowski é Professor Associado do Departamento de Informática da PUC-Rio, onde orienta pesquisas de mestrado e doutorado nas áreas de Engenharia de Software e Ciência de Dados e coordena projetos de pesquisa e desenvolvimento junto a diversas empresas. É doutor e mestre em Engenharia de Sistemas e Computação na área de Engenharia de Software e bacharel em Ciência da Computação, todos pela UFRJ. É bolsista de produtividade em pesquisa do CNPq e possui mais de 150 artigos publicados nos principais veículos da sua área de atuação.

Capítulo 1
Introdução

Bem-vindo(a) ao incrível mundo da Estatística aplicada à Ciência de Dados! Você está prestes a embarcar em uma jornada fascinante que o(a) levará a desvendar os segredos por trás dos números, a compreender o poder das probabilidades e a dominar as ferramentas essenciais para extrair conhecimento valioso a partir dos dados.

A Estatística é a linguagem que nos permite decifrar o que os números têm a dizer e transformá-los em *insights* que impulsionam a tomada de decisões informadas. Neste capítulo inicial, vamos explorar por que estudar Estatística é fundamental para qualquer cientista de dados e estabelecer as bases para sua jornada de aprendizado. Tudo pronto para mergulhar nesse universo emocionante? Vamos lá!

1.1 Por que estudar Estatística para Ciência de Dados?

No nosso dia a dia, frequentemente nos deparamos com a necessidade de tomar decisões importantes, muitas vezes baseadas em informações incompletas e permeadas por graus variáveis de incerteza. Esta realidade não é exclusiva das escolhas pessoais, mas também se estende ao cenário empresarial, onde a tomada de decisões estratégicas é um componente crítico para o sucesso das organizações.

É nesse contexto que profissionais de Ciência de Dados desempenham um papel fundamental, aplicando um conjunto de competências e ferramentas para abordar com eficácia a incerteza subjacente a essas decisões. Elas não apenas reconhecem a

importância de gerenciar a incerteza, mas também compreendem como quantificá-la e utilizá-la como uma vantagem estratégica. Por meio da aplicação de métodos estatísticos, os cientistas de dados têm a habilidade de traduzir dados em informações valiosas, transformando o território complexo da incerteza em uma área de atuação clara e informada. De fato, é muitas vezes na incerteza que surgem as oportunidades mais significativas, e a Ciência de Dados é o guia confiável para explorar esse terreno desafiador.

A Probabilidade, por sua vez, é um subcampo fundamental da Estatística com uma história centenária, desempenhando um papel preponderante na abordagem das incertezas e variações inerentes às tomadas de decisão em nossa rotina. Esta disciplina oferece uma estrutura teórica sólida, uma terminologia especializada e uma série de métodos formais que se mostram essenciais para enfrentar tais desafios com eficácia. Ao adentrar o campo da Ciência de Dados, torna-se evidente que os métodos probabilísticos fornecem a base sobre a qual nossos modelos e análises são construídos. Além disso, à medida que exploramos dados, percebemos que muitos dos conceitos primordiais que guiam nossa compreensão têm suas raízes firmemente estabelecidas no domínio da Probabilidade. Estamos constantemente em busca de compreender a média ou o valor mais provável de um evento, bem como examinar como diferentes resultados podem se distribuir ao longo de um espectro de possibilidades. Consequentemente, à medida que embarcamos nesta jornada de exploração e descoberta, deparamo-nos com a Estatística e a Probabilidade como ferramentas inestimáveis que nos permitem decifrar e alavancar o mundo que nos cerca.

Ao adentrar o mundo da Ciência de Dados, a escolha da linguagem de programação desempenha um papel crítico na jornada do cientista de dados. Em nossa busca por uma linguagem que ofereça flexibilidade, poder computacional e uma ampla variedade de bibliotecas estatísticas, destacamos duas opções notáveis: Python e R. Ambas as linguagens possuem suas próprias vantagens distintas, tornando-as escolhas populares na comunidade de Ciência de

Dados. O Python, conhecido por sua sintaxe limpa e versatilidade, oferece um ambiente ideal para desenvolver modelos de Machine Learning, implementar algoritmos de aprendizado de máquina e manipular dados com facilidade. Enquanto isso, o R, desenvolvido especificamente para análise estatística e visualização de dados, se destaca na criação de gráficos informativos e na execução de análises estatísticas complexas. Neste livro, optamos por combinar o poder dessas duas linguagens para fornecer uma base sólida e abrangente para a Estatística aplicada à Ciência de Dados. A integração dessas linguagens permite que os leitores e leitoras explorem as funcionalidades únicas de cada uma, capacitando-os a abordar diversos aspectos da análise de dados e da modelagem estatística com confiança.

Os conceitos-chave apresentados anteriormente desempenham um papel central no campo da Estatística para Ciência de Dados e são fundamentais para uma compreensão sólida e abrangente deste tópico. Eles serão minuciosamente explorados e explicados ao longo deste livro, fornecendo a você as ferramentas essenciais necessárias para enfrentar os desafios complexos e emocionantes da Ciência de Dados. Os temas incluem a análise criteriosa de dados, reconhecimento das distribuições de probabilidade subjacentes, estimação de parâmetros importantes e a aplicação desses conceitos na construção e avaliação de modelos de Machine Learning. Além disso, os leitores e leitoras aprenderão a quantificar incertezas, medir o desempenho preditivo e selecionar métricas adequadas para avaliar seus modelos, tudo dentro do contexto dinâmico e interdisciplinar da Ciência de Dados.

1.2 Conceitos fundamentais

Nesta seção apresentaremos alguns conceitos fundamentais de Estatística e Ciência de Dados.

Estatística

A estatística é um conjunto de técnicas que permite de forma sistemática organizar, descrever, analisar e interpretar dados advindos de diversas origens a fim de extrair conclusões. Pode ser subdivida em três grandes áreas:

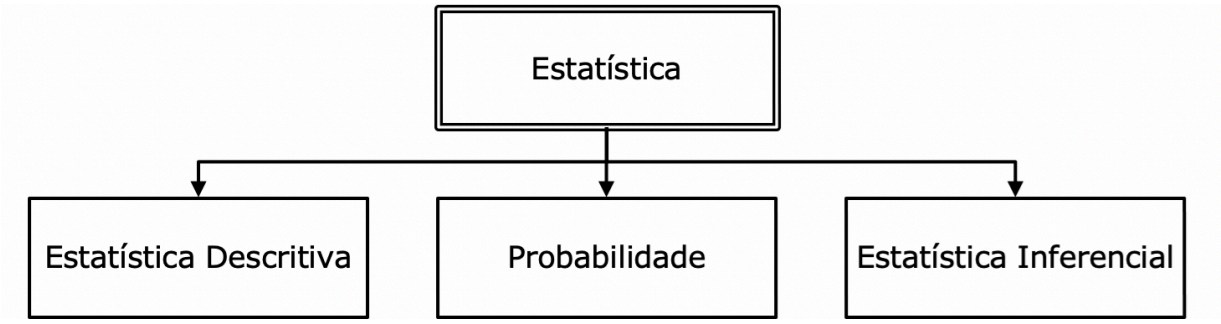


Figura 1.1: Áreas da Estatística.

Estatística Descritiva

É o ramo da Estatística que se ocupa em organizar e descrever os dados, que podem ser expressos em tabelas e gráficos. A Estatística Descritiva é muito utilizada na etapa de análise exploratória de dados de projetos de Ciência de Dados e pode ser dividida em dois grupos:

- **Medidas de posição / Medidas de tendência central:** permitem saber a tendência de concentração dos dados e o melhor número que a representa. Exemplos: média, moda, mediana e medidas separatrizes.
- **Medidas de dispersão:** permitem saber o grau de dispersão dos dados em torno de uma medida de posição, geralmente a média dos dados. Exemplos: desvio padrão, variância e coeficiente de variação.

Abordaremos a Estatística Descritiva em mais detalhes nos capítulos 2 e 3 deste livro.

Probabilidade

Permite-nos descrever os fenômenos aleatórios, ou seja, aqueles em que está presente a incerteza. A probabilidade mede o quão provável é a ocorrência de um evento, como a probabilidade de ocorrência de uma face "cara" no lançamento de uma moeda ou de um empregado pedir demissão. A probabilidade de um evento de interesse pode ser calculada dividindo a contagem de todas as ocorrências deste evento pelo total de possibilidades deste evento. Assim:

Probabilidade(evento = X) = ocorrências quando evento é X/(ocorrências quando evento é X + ocorrências quando evento não é X)

A probabilidade é sempre um valor decimal entre 0 e 1 que pode ser convertido em um percentual multiplicando este valor por 100, em que 0 representa nenhuma probabilidade (0%) e 1 representa a probabilidade total (100%). Se todas as possíveis ocorrências de um evento são igualmente prováveis (tal como o resultado do lançamento de uma moeda não viciada ser "cara" ou "coroa"), a probabilidade de cada ocorrência é 1 dividido pelo total de possíveis ocorrências. No caso da moeda, esta probabilidade é 1 em 2, sendo representada numericamente por 1/2 ou 50%.

O complemento de uma probabilidade pode ser calculado como 1 menos a probabilidade da ocorrência de um evento. No exemplo da moeda:

probabilidade de coroa = 1 - P(cara)

Abordaremos mais conceitos sobre o cálculo de probabilidades nos capítulos 4 e 5 deste livro.

Inferência Estatística

É o estudo de técnicas que possibilitam a extrapolação, a um grande conjunto de dados, das informações e conclusões obtidas a partir da amostra. Abordaremos mais conceitos sobre Inferência Estatística, incluindo noções de amostragem e reamostragem e modelagem estatística usando Machine Learning nos capítulos 6 e 7 deste livro. Além disso, alguns conceitos de Inferência Estatística são empregados no contexto de Ciência de Dados na experimentação contínua, abordada no capítulo 8.

Ciência de Dados

A figura a seguir ilustra o esquema básico de um projeto de Ciência de Dados (proposto originalmente no livro *Introdução a Data Science: Algoritmos de Machine Learning e métodos de análise*, publicado pela editora Casa do Código), que pode ser resumido em 7 etapas.

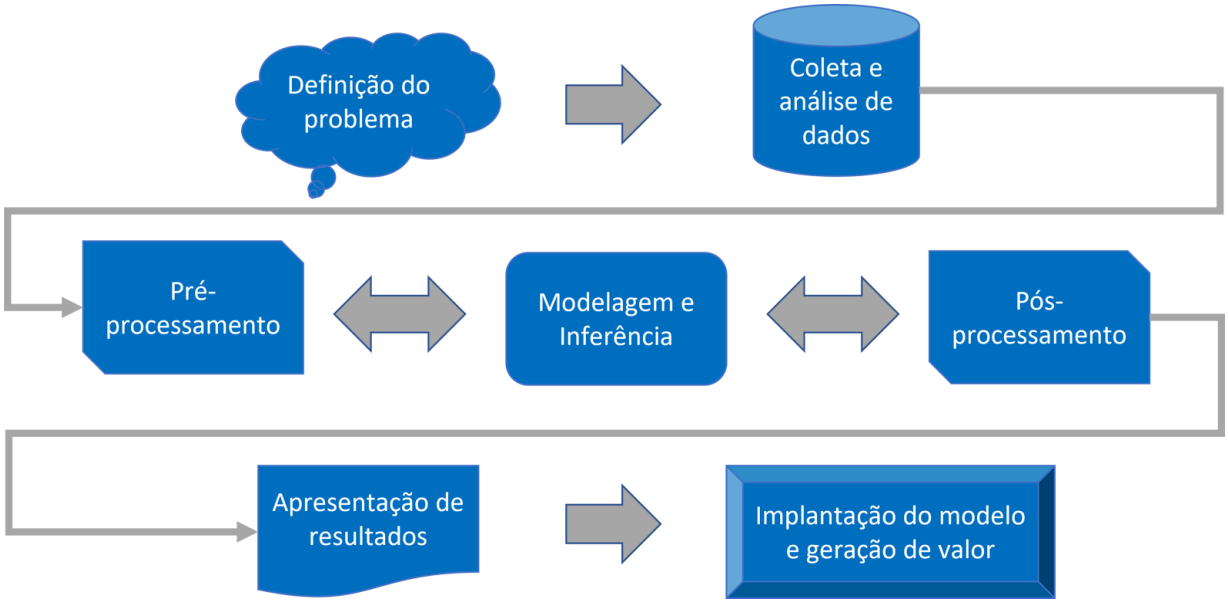


Figura 1.2: Esquema básico de um projeto de Ciência de Dados

Um projeto de Ciência de Dados começa com um **problema, necessidade ou ideia**. Nesta etapa inicial, deve-se primeiro ter em mente o problema que se deseja resolver, e, em seguida, os objetivos devem ser definidos, bem como elencar as perguntas que

as pessoas gestoras desejam responder. A segunda etapa consiste em levantar as informações necessárias e efetivamente coletar os dados para resolver os problemas levantados na etapa anterior. Nesta etapa é realizada a **análise exploratória de dados** que busca, entre outras atividades, identificar dados ruidosos, incompletos ou incorretos. Exemplos de problemas que podemos encontrar nos dados na etapa de análise exploratória são:

- dados que tiveram o preenchimento realizado de forma incorreta;
- dados incompletos, com campos em branco;
- dados inconsistentes, por exemplo, campos com letras onde era esperado um número;
- dados insuficientes ou pouco representativos, por exemplo, coletados apenas em um bairro quando precisamos analisar toda uma cidade.

A Estatística nos oferece diversas ferramentas e técnicas para analisarmos com cuidado os nossos dados e nos ajudar a identificar problemas e possibilitar o tratamento adequado destes dados na etapa seguinte, de **pré-processamento**. Existem diversas técnicas que podem ser utilizadas para tratar estes e outros problemas encontrados nos dados, tais como a limpeza (exclusão de linhas ou colunas), preenchimento de valores faltantes (usando, por exemplo, a média, moda ou mediana dos dados) ou mesmo o enriquecimento de dados (obtenção de dados a partir de outras fontes ou novas coletas).

A etapa de pré-processamento costuma ser a etapa mais demorada e trabalhosa do projeto de Ciência de Dados (em geral, consome cerca de 70% do tempo total do projeto) e é fundamental para a etapa seguinte, **modelagem e inferência**, onde é realizada a modelagem preditiva, caso o objetivo do projeto seja a construção de um modelo de Machine Learning. Um pré-processamento adequado é importante pois, quanto mais "sujos" forem os nossos dados, menos confiável será o resultado do modelo construído utilizando estes dados (*"gargabe in, garbage out"*). Nesta etapa,

elencamos os modelos possíveis e passíveis para cada tipo de problema, estimamos os hiperparâmetros que compõem os modelos e avaliamos os resultados de cada modelo, usando métricas e um processo justo de comparação.

Vale a pena ressaltar que um modelo de Machine Learning sempre terá algum erro, como reforça a famosa frase do estatístico inglês George Box: "Todos os modelos estão errados, mas alguns são úteis". Os erros podem significar predições imperfeitas por conta dos problemas nos dados que mencionamos anteriormente ou mesmo por problemas na modelagem (por exemplo, utilizando um modelo simples demais para o problema) ou até mesmo pela própria natureza da modelagem como forma de representação reduzida da realidade. Ainda assim, apesar das incertezas, na maioria das vezes é mais importante construir um modelo que seja bom o suficiente - apesar de eventualmente cometer alguns erros - do que não termos nenhum modelo.

A seguir, temos a etapa de **pós-processamento**, na qual combinamos as heurísticas de negócio com os modelos ajustados na etapa de anterior e fazemos uma avaliação final, tendo em vista os pontos fortes e dificuldades encontradas na implementação de cada um dos modelos. Na sequência, a **apresentação de resultados** visa relatar para o cliente ou *stakeholders* a metodologia adotada para endereçar a solução, comparar os resultados do melhor modelo encontrado com o *benchmark* atual (caso haja) e planejar os passos para a implementação da solução de proposta. Finalmente, na etapa de **implantação do modelo e geração de valor**, implementa-se o modelo em produção, buscando a geração de valor para o negócio.

1.3 Estatística e Ciência de Dados

A sinergia entre a Estatística e a Ciência de Dados se manifesta diretamente nas etapas anteriormente apresentadas, desde a análise exploratória inicial até o pós-processamento dos resultados.

Essa colaboração interdisciplinar enriquece as abordagens e melhora os resultados em cada etapa.

Na **análise exploratória de dados**, a Estatística fornece métodos para resumir e visualizar os dados, identificando padrões, anomalias, distribuições e correlações. Já a Ciência de Dados utiliza ferramentas computacionais para manipular grandes conjuntos de dados e aplicar visualizações complexas, incluindo gráficos interativos e mapas de calor, facilitando uma compreensão mais profunda dos dados.

No **pré-processamento de dados**, a Estatística auxilia na identificação e tratamento de valores ausentes, *outliers* e na transformação de variáveis para atender aos pressupostos de modelos estatísticos. Além disso, oferece métodos robustos para normalização e padronização. A Ciência de Dados, por sua vez, emprega algoritmos e técnicas para limpeza e preparação de dados em escala, incluindo a codificação de variáveis categóricas, imputação de dados faltantes e seleção de características, utilizando bibliotecas de software.

Na **modelagem e inferência**, a Estatística oferece a fundamentação teórica para construir modelos que possam fazer inferências sobre os dados. Além disso, no contexto da experimentação contínua, permite comparar modelos utilizando testes de hipótese. A Ciência de Dados aplica técnicas e algoritmos de Machine Learning para explorar padrões nos dados e fazer predições.

No **pós-processamento** a Estatística contribui com métodos para avaliar a precisão e a confiabilidade dos resultados dos modelos, através de métricas de desempenho, intervalos de confiança e testes de significância. A Ciência de Dados pode ser empregada neste ponto para integrar técnicas de interpretabilidade de modelos e

explicabilidade para fornecer *insights* sobre como os modelos fazem suas predições.

Ou seja, é possível perceber como a Estatística e a Ciência de Dados se complementam para melhorar a qualidade e a precisão dos *insights* gerados. Enquanto a Estatística oferece o rigor metodológico e as bases para inferências confiáveis, a Ciência de Dados expande as possibilidades com técnicas computacionais avançadas e modelos complexos, capazes de lidar com o volume, a velocidade e a variedade dos dados modernos. Juntas, essas disciplinas permitem resolver problemas complexos em um mundo cada vez mais orientado por dados.

CAPÍTULO 2
Estatística Descritiva: Conceitos básicos, tipos de variáveis e gráficos

Vimos no capítulo anterior que a Estatística Descritiva é focada em organizar e descrever os dados, que podem ser expressos em tabelas e gráficos, sendo muito utilizada na etapa de análise exploratória de dados de projetos de Ciência de Dados. Ela pode ser dividida em dois grupos:

- **Medidas de posição / Medidas de tendência central:** permitem saber a tendência de concentração dos dados e o melhor número que a representa. Exemplos: média, moda, mediana e medidas separatrizes.
- **Medidas de dispersão:** permitem saber o grau de dispersão dos dados em torno de uma medida de posição, geralmente a média dos dados. Exemplos: desvio padrão, variância e coeficiente de variação.

Este capítulo abordará conceitos básicos de Estatística Descritiva, tipos de variáveis e gráficos.

2.1 Conceitos básicos

Em muitos casos, precisamos de amostras da população para realizar uma pesquisa, uma vez que ela pode possuir limitações financeiras, inacessibilidade a alguns elementos da população, ou até mesmo a subjetividade do caso concreto envolvido. Imagine que você precisa avaliar se está com anemia, por meio do exame de sangue, e em vez de uma amostra do seu sangue o médico usa a

população do seu sangue? Isso não é viável, não é mesmo? Por isso se faz necessária a retirada de uma amostra.

População: é a coleção de todos os indivíduos que possuem determinadas características, as quais estamos interessados em estudar. Representamos por: *N = Tamanho populacional*

Amostra: é um subconjunto da população, uma parte dos indivíduos que possuem determinadas características. Representamos por: *n = Tamanho amostral*

> Observação: sempre que falarmos em amostra, usaremos iniciais minúsculas e, quando tratarmos de População, por sua vez, iniciais maiúsculas.

Censo: exame de todos os elementos da população.

Dados brutos: são dados na sua forma mais primitiva, desprovidos de ordenação, assim que coletados.

Rol estatístico: são os dados brutos já ordenados, em ordem crescente ou decrescente.

Dados tabelados (ou tabulados): os dados podem ser expressos em tabelas de frequência, tanto em Frequência Absoluta Simples (f_i), Frequência Relativa Simples (fr), Frequência Absoluta Acumulada (f_{ac}) ou Frequência Relativa Acumulada (fr_{ac}).

Vejamos exemplos de cada uma delas:

Frequência Absoluta Simples (f_i): é a contagem simples de elementos.

Idade(Anos)	**f_i**
10	4

Idade(Anos)	f_i
30	8
50	4
70	3
90	1

Frequência Relativa Simples (fr): é a contagem simples de elementos, divididos pela soma das frequências simples, ou seja, representa a proporção ou o percentual de observações.

Idade(Anos)	fr
10	4/20 = 0,2 ou 20%
30	8/20 = 0,4 ou 40%
50	4/20 = 0,2 ou 20%
70	3/20 = 0,15 ou 15%
90	1/20 = 0,05 ou 5%

Frequência Absoluta Acumulada (f_{ac}): é a contagem acumulada até a classe de interesse (inclusive).

Idade(Anos)	f_{ac}
10	4
30	(8 + f1) = 12
50	(4 + f1 + f2) = 16
70	(3 + f1 + f2 + f3) = 19
90	(1 + f1 + f2 + f3 + f4) = 20

Frequência Relativa Acumulada (fr_{ac}): é a contagem acumulada até a classe de interesse, divididos pela soma das frequências simples.

Idade(Anos)	fr_{ac}
10	4/20 = 0,2 ou 20%
30	12/20 = 0,6 ou 60%
50	16/20 = 0,8 ou 80%
70	19/20 = 0,95 ou 95%
90	20/20 = 1 ou 100%

Distribuições de frequência em classes: quando possuímos um grande conjunto de dados, se agruparmos em classes, teremos uma boa ideia do comportamento dos dados.

Observação: as amplitudes dos intervalos de classe (limite superior – limite inferior), não necessariamente devem ser iguais.

Vejamos exemplos:

Idade(Anos)	f_i
2 ⊣ 7	4
7 ⊣ 12	8
12 ⊣ 17	4
17 ⊣ 22	3
22 ⊣ 27	1

Observação: a notação ⊣ significa que o 7, que é o limite superior da primeira classe, está contido no intervalo. Já o 2, que é o limite inferior da primeira classe, não está contido.

2.2 Variáveis e tipos de variáveis

Variável é qualquer característica associada a uma população. Podem ser classificadas em:

Variáveis qualitativas: representam atributos, qualidades, que podem ser divididas em dois grupos, ordinais (quando existir uma ordem implícita) e nominais (quando não existir uma ordem implícita). Exemplos:

- **Variáveis qualitativas ordinais:** classe social, grau de instrução e estágio da doença.
- **Variáveis qualitativas nominais:** sexo, cor dos olhos, fumante/não fumante e doente/sadio.

Observação: cabe destacar que existe um tipo de variável qualitativa cuja quantificação é muito útil, a chamada variável dicotômica (assume o valor 0 ou 1). Para essa variável, só podem ocorrer duas realizações, que chamamos de fracasso e sucesso. A variável fumante/não fumante é um exemplo disso (só pode assumir dois resultados possíveis).

Variáveis quantitativas: podem ser divididas em dois grupos, discretas (quando for finita e enumerável) e contínuas (quando os resultados possíveis pertencerem a um intervalo de números reais e resultados de mensuração). Exemplos:

- **Variáveis quantitativas discretas:** número de filhos, número de carros e número de cigarros fumados por dia.
- **Variáveis quantitativas contínuas:** peso, altura e salário.

A figura a seguir resume os tipos de variáveis:

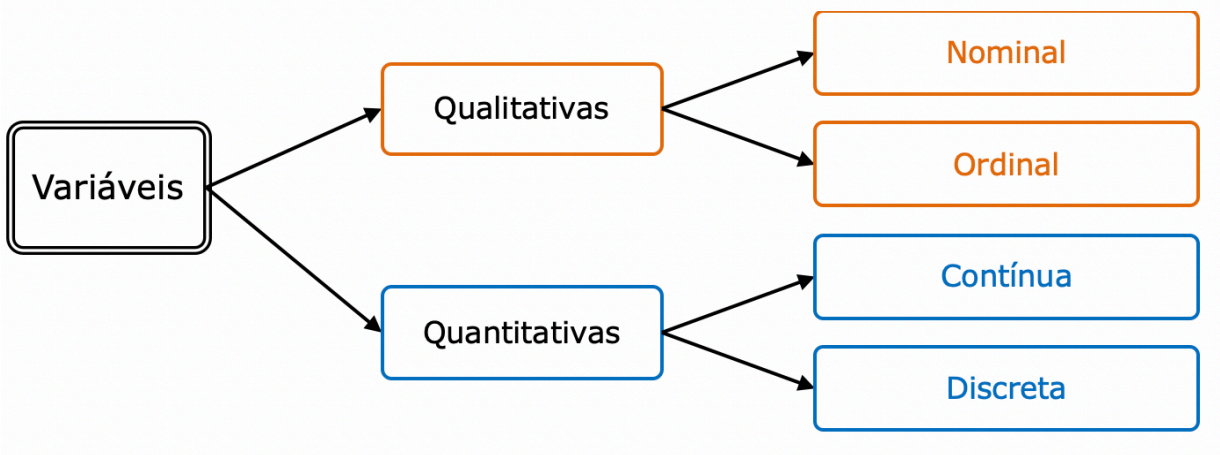


Figura 2.1: Tipos de variáveis.

2.3 Gráficos adequados a cada tipo de variável

A importância dos gráficos se faz presente quando queremos observar o comportamento de determinadas variáveis de forma rápida e sintetizada, especialmente quando possuem um número elevado de observações. Nosso cérebro entende melhor gráficos que números soltos. Contudo, existem gráficos apropriados para cada tipo de variável e vamos estudá-los nos tópicos a seguir.

Uma das bibliotecas mais conhecidas para a construção de gráficos em Python é a *Matplotlib*. Ela inclui o módulo *pyplot*, que será utilizado nos exemplos de gráficos a seguir. Como o intuito deste livro não é nos aprofundarmos nas bibliotecas existentes, mostraremos exemplos simples, porém, é possível customizar os

gráficos usando opções mais avançadas. Para mais informações, consulte a documentação da biblioteca Matplotlib, em https://matplotlib.org/. Você também pode explorar outras bibliotecas bem conhecidas para a construção de gráficos em Python, como a *Seaborn*, *Ploty* e *ggplot*.

Você pode digitar os códigos dos exemplos a seguir na IDE de sua preferência, ou baixá-los diretamente de https://github.com/tatianaesc/estatisticadatascience.

Gráficos para variáveis qualitativas

Dentre os gráficos para variáveis qualitativas, podemos destacar dois:

- **Gráfico de setores ou pizza:** destina-se a apresentar a composição, usualmente em porcentagem, das partes de um todo. Consiste em um raio arbitrário, representando o todo, dividido em setores (fatias da pizza), que correspondem às partes de maneira proporcional.

Um gráfico de pizza pode ser criado em Python chamando a função `pie()` e passando como parâmetros os valores e os nomes das categorias. O exemplo a seguir cria um conjunto de dados com três categorias e, para cada uma delas, um valor inteiro associado. Digite o trecho de código a seguir e veja o resultado:

```
# Imports de bibliotecas
from matplotlib import pyplot
# Categorias e tamanhos dos setores. Os setores serão ordenados e plotados no sentido anti-horário
categorias = 'Java', 'R', 'Python'
setores = [45, 25, 30]
# Criação do gráfico
pyplot.pie(x=setores, labels=categorias)
pyplot.show()
```

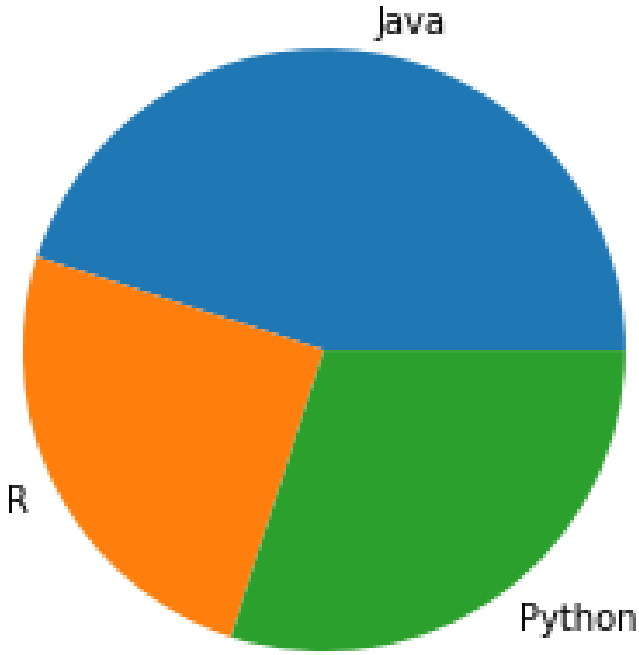


Figura 2.2: Gráfico de pizza construído em Python.

Um gráfico de pizza pode ser criado em R chamando a função `plot_ly()` e passando como parâmetros os dados, valores e os nomes das categorias, dizendo o tipo de gráfico no **type="pie"**. O exemplo a seguir cria um conjunto de dados com três categorias e, para cada uma delas, um valor em porcentagem associado. Digite o trecho de código a seguir e veja o resultado:

```
install.packages("plotly")
library(plotly)
# Categorias, tamanhos dos setores e %. Os setores serão ordenados e
plotados no sentido anti-horário
categorias = c("Java", "R", "Python")
setores = c(45, 25, 30)
# Criando o data.frame (Resultado do cruzamento de categorias e setores)
dados=data.frame(categorias,setores)
# Criação do Gráfico e Exibição
plot_ly(dados,
        labels = ~categorias,
```

```
values = ~setores,
type = 'pie')
```

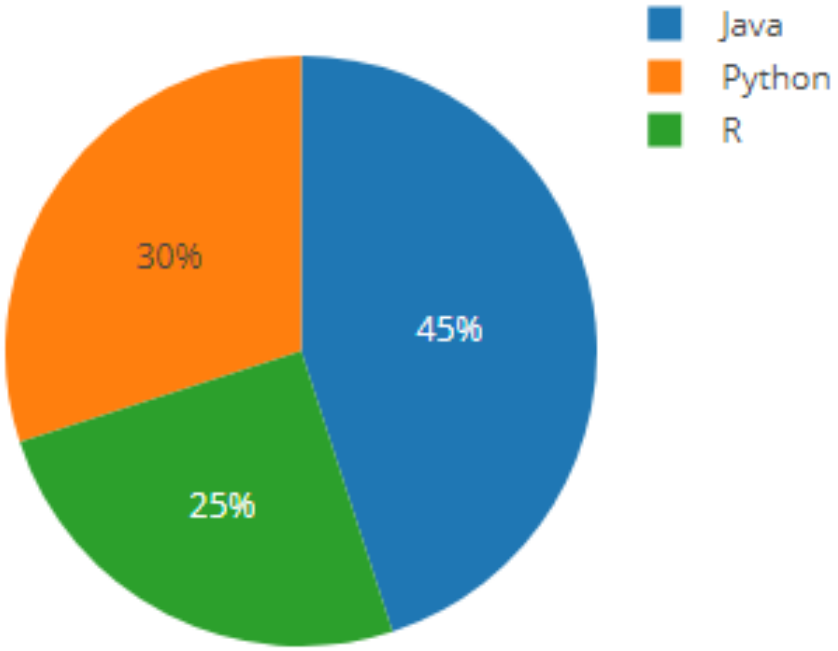


Figura 2.3: Gráfico de pizza construído em R.

- **Gráfico de barras/colunas:** consiste em construir retângulos ou barras, em que uma das dimensões é proporcional à magnitude a ser representada, sendo a outra arbitrária, contudo igual para todas as barras. Essas barras são dispostas paralelamente às outras, horizontal ou verticalmente.

> Observação: em algumas referências é chamado de gráfico de barras somente aquele que apresenta as barras dispostas horizontalmente, sendo chamado de gráfico de colunas quando disposto verticalmente. O gráfico de colunas é preferível quando há um número grande de categorias, facilitando a leitura.

Um gráfico de colunas pode ser criado em Python chamando a função `bar()` e passando como parâmetros os nomes das categorias para o eixo x e os valores para o eixo y. O exemplo a seguir cria um

conjunto de dados com três categorias e, para cada uma delas, um valor inteiro associado. Digite o trecho de código a seguir e veja o resultado:

```
# Imports de bibliotecas
from matplotlib import pyplot
# Categorias
x = ['R', 'Java', 'Python']
# Valores de cada categoria
y = [27, 42, 89]
# Criação do gráfico
pyplot.bar(x, y)
# Exibição do gráfico
pyplot.show()
```

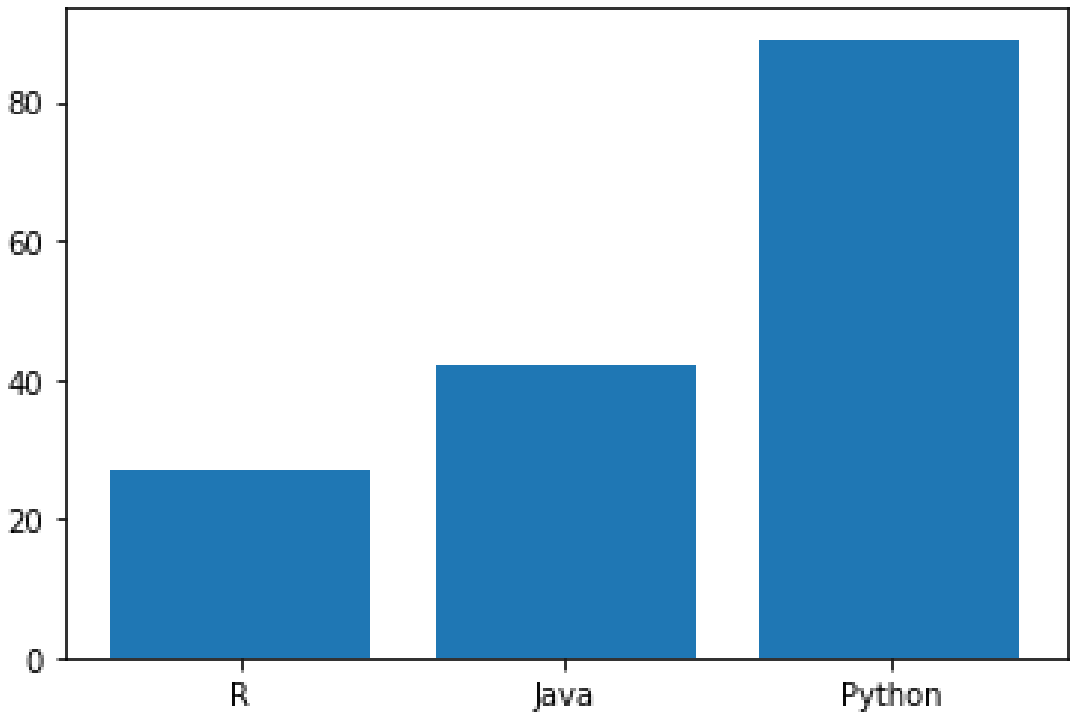


Figura 2.4: Gráfico de Colunas construído em Python.

Um gráfico de colunas pode ser criado em R chamando a função `barplot()` e passando como parâmetros os valores para o eixo x e as categorias eixo y, por meio do parâmetro `names.arg`. O exemplo a seguir cria um conjunto de dados com três categorias e, para cada

uma delas, um valor inteiro associado. Digite o trecho de código a seguir e veja o resultado:

```
# Categorias
x = c("Java", "R", "Python")
# Valores de cada categoria
y = c(27, 42, 89)
# Criando o data.frame (Resultado do cruzamento de categorias e valores)
dados=data.frame(x,y)
# Criação do Gráfico e Exibição
barplot(dados$y, names.arg=dados$x,col="light blue")
```

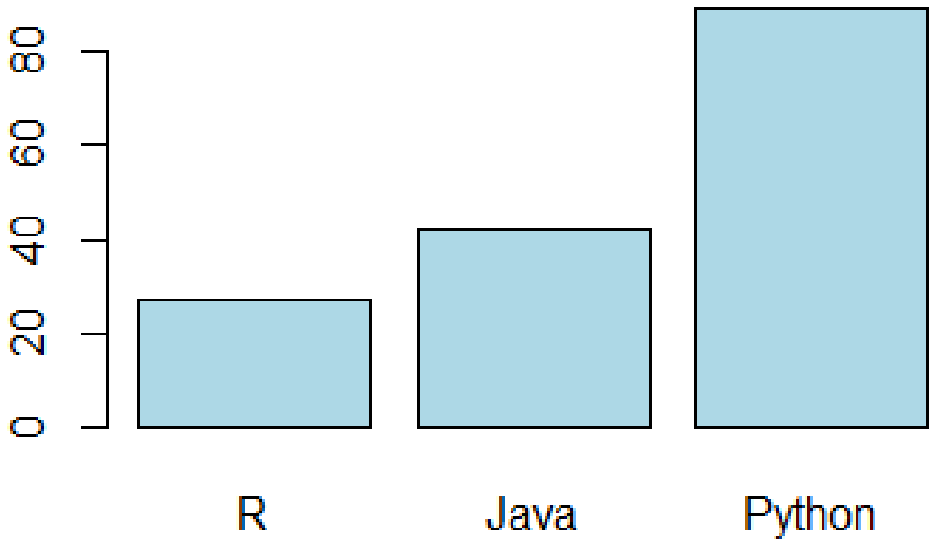


Figura 2.5: Gráfico de Colunas construído em R.

Se quiser fazer o mesmo gráfico do exemplo acima usando barras em Python, basta utilizar a função `barh()`. Experimente:

```
# Imports de bibliotecas
from matplotlib import pyplot
# Categorias
x = ['R', 'Java', 'Python']
# Valores de cada categoria
y = [27, 42, 89]
# Criação do gráfico
pyplot.barh(x, y)
```

```
# Exibição do gráfico
pyplot.show()
```

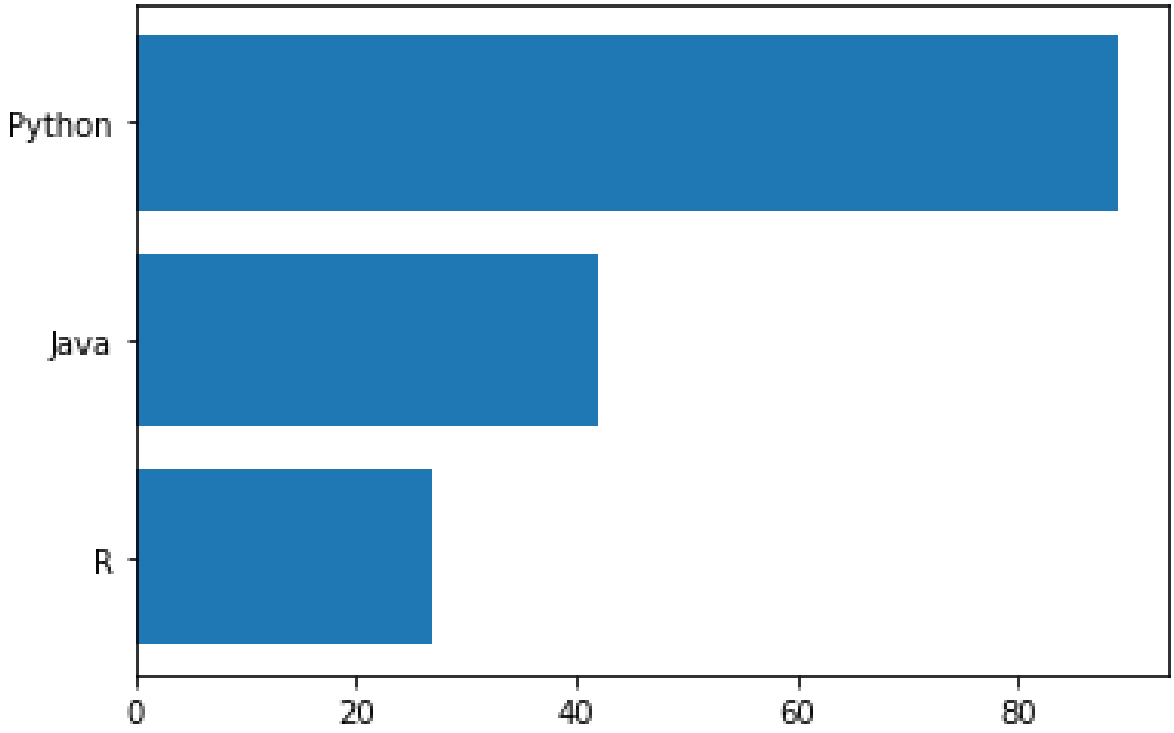


Figura 2.6: Gráfico de Barras construído em Python.

Se quiser fazer o mesmo gráfico do exemplo acima usando barras em R, basta utilizar o parâmetro `horiz = T` da mesma função `barplot()`:

```
# Categorias
x = c("Java", "R", "Python")
# Valores de cada categoria
y = c(27, 42, 89)
# Criando o data.frame (Resultado do cruzamento de categorias e valores)
dados=data.frame(x,y)
# Criação do Gráfico e Exibição
barplot(dados$y, names.arg=dados$x,col="light blue",horiz = T)
```

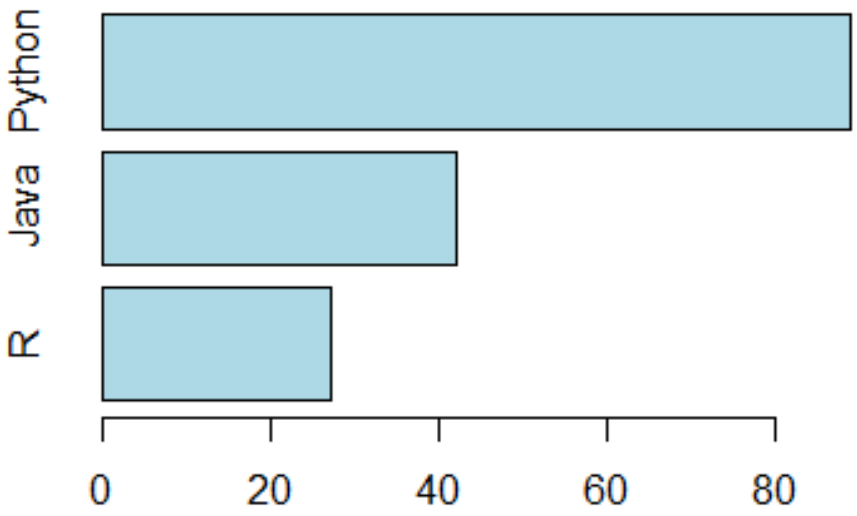


Figura 2.7: Gráfico de Barras construído em R.

Gráficos para variáveis quantitativas

Teremos uma maior variedade de gráficos para variáveis quantitativas. No caso de variáveis **quantitativas discretas**, geralmente organizamos os dados em tabelas de frequência e a representação gráfica é mediante gráfico de barras, similar ao visto para as variáveis qualitativas, também podendo ser representadas por gráficos de dispersão:

- **Gráfico de dispersão:** parecido com o gráfico de barras, porém são plotadas somente as intersecções dos pontos de x e y.

Um gráfico de dispersão (em inglês, *scatter plot*) pode ser criado em Python chamando a função `scatter()` e passando como parâmetros os valores de x e y. O exemplo a seguir apresenta apenas cinco pares de valores (x,y). Digite o trecho de código a seguir e veja o resultado:

```
# Imports de bibliotecas
from matplotlib import pyplot
# Valores de x
```

```
x = [46, 9, 25, 42, 95]
# Valores de y
y = [12, 23, 4, 9, 15]
# Criação do gráfico
pyplot.scatter(x, y)
# Exibição do gráfico
pyplot.show()
```

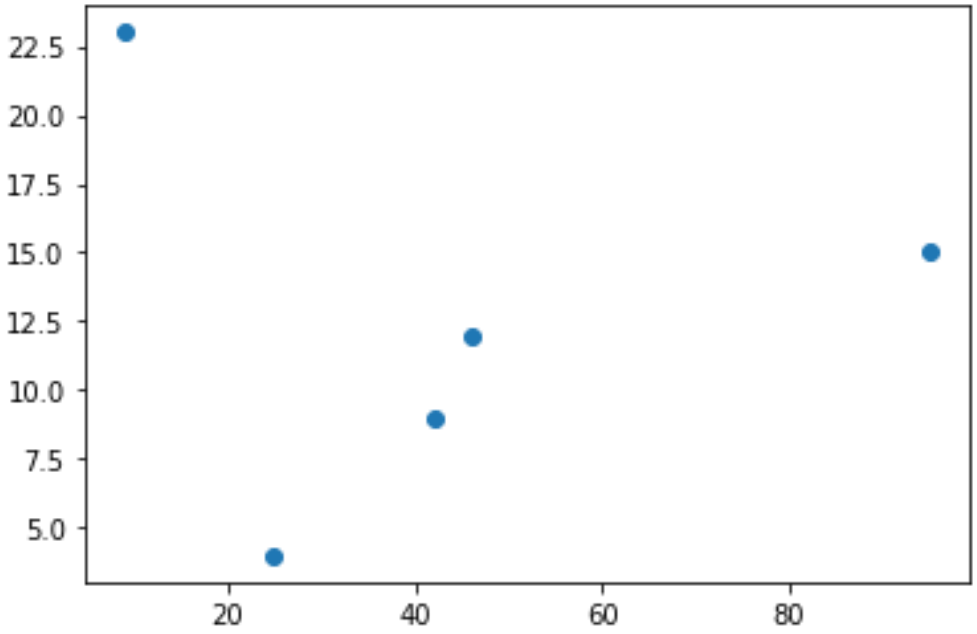


Figura 2.8: Gráfico de dispersão construído em Python.

Elementos utilizados no gráfico `ggplot` :

1. Base de dados que será utilizada;
2. *Aesthetic mapping* (a parte estética do gráfico: eixos, cores, tamanhos, textos) - depende da geometria;
3. Geometria que será utilizada (tipo de gráfico: dispersão, barras, histograma...).

Um gráfico de dispersão pode ser criado em R chamando a função `ggplot()` e passando como parâmetros os dados, os eixos x e y e a geometria de interesse: `geom_point()` . O exemplo a seguir apresenta apenas cinco pares de valores (x,y). Digite o trecho de código a seguir e veja o resultado:

```
# Valores de x
x = c(46, 9, 25, 42, 95)
# Valores de y
y = c(12, 23, 4, 9, 15)
#
dados=data.frame(x,y)
# Criação do Gráfico e Exibição
ggplot(dados, aes(x, y)) +
      geom_point(size=3,col="blue")
```

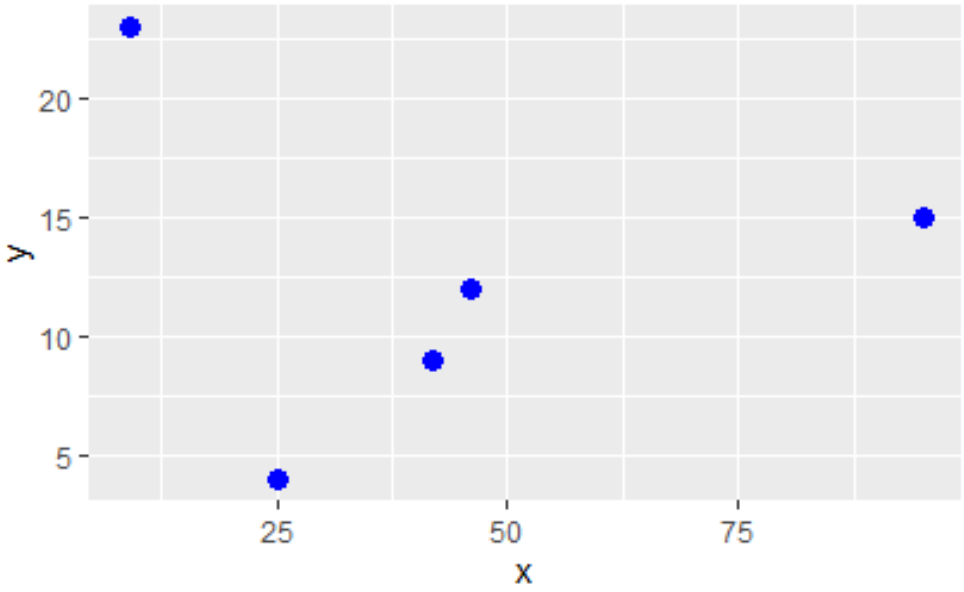


Figura 2.9: Gráfico de dispersão construído em R.

Já para as **quantitativas contínuas**, se exibirmos os dados em gráfico de barras, muitas informações seriam perdidas. Desta forma, podemos construir uma tabela de *Distribuições de frequência em classes*, conforme já apresentado anteriormente:

Idade(Anos)	$\mathbf{f_i}$
2 ⊣ 7	4
7 ⊣ 12	8
12 ⊣ 17	4

Idade(Anos)	f_i
17 ⊣ 22	3
22 ⊣ 27	1

Para representar essa tabela em forma gráfica, faz-se necessário o uso de um gráfico chamado Histograma:

- **Histograma:** um gráfico de barras contíguas, com as bases proporcionais aos intervalos de classe e a área do retângulo proporcional à respectiva frequência.

Um histograma pode ser criado em Python chamando a função `hist()` e passando como parâmetros os valores de x e y. O exemplo a seguir apresenta apenas cinco pares de valores (x,y). Digite o trecho de código a seguir e veja o resultado:

```
# Imports de bibliotecas
from matplotlib import pyplot
# Valores
x = [64, 77, 45, 64, 88, 11, 22, 67, 81, 88, 43, 51, 18, 32, 3, 10, 90,
20, 24, 63, 51, 96, 11, 78]
# Número de intervalos (bins)
num_bins = 6
# Criação do gráfico
pyplot.hist(x, num_bins)
# Exibição do gráfico
pyplot.show()
```

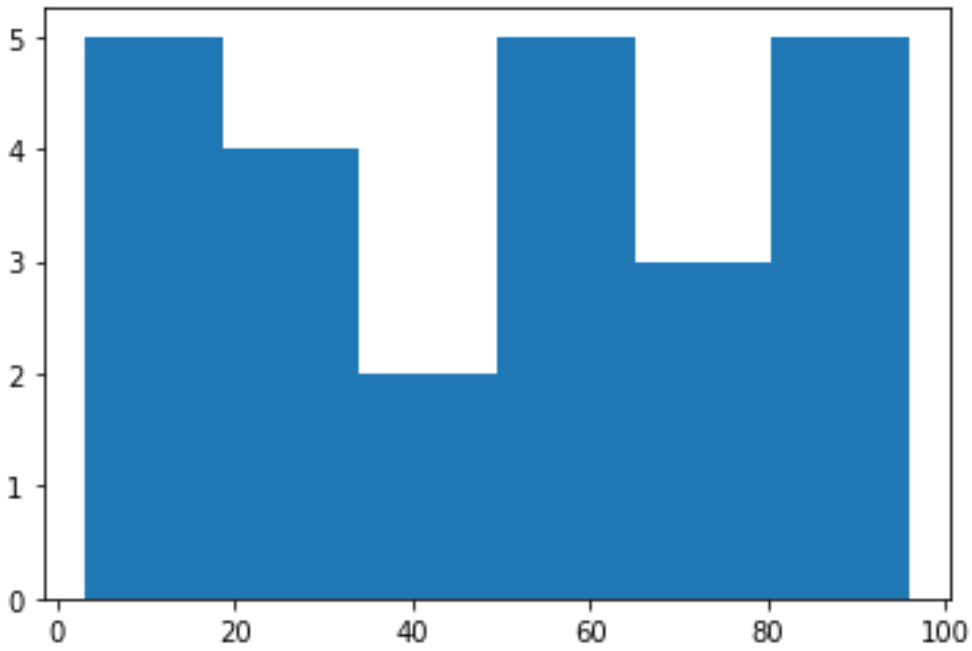


Figura 2.10: Histograma construído em Python.

Um histograma pode ser criado em R chamando a função `ggplot()` e passando como parâmetros os dados, eixos x e a geometria de interesse: `geom_histogram()` . Digite o trecho de código a seguir e veja o resultado:

```
library(ggplot2)
# Valores
x = c(64, 77, 45, 64, 88, 11, 22, 67, 81, 88, 43, 51, 18, 32, 3, 10, 90,
20, 24, 63, 51, 96, 11, 78)
dados=data.frame(x)
# Número de intervalos (bins)
num_bins = 6
# Criação do Gráfico e Exibição
ggplot(dados,aes(x)) +
  geom_histogram(bins = num_bins,fill="light blue")
```

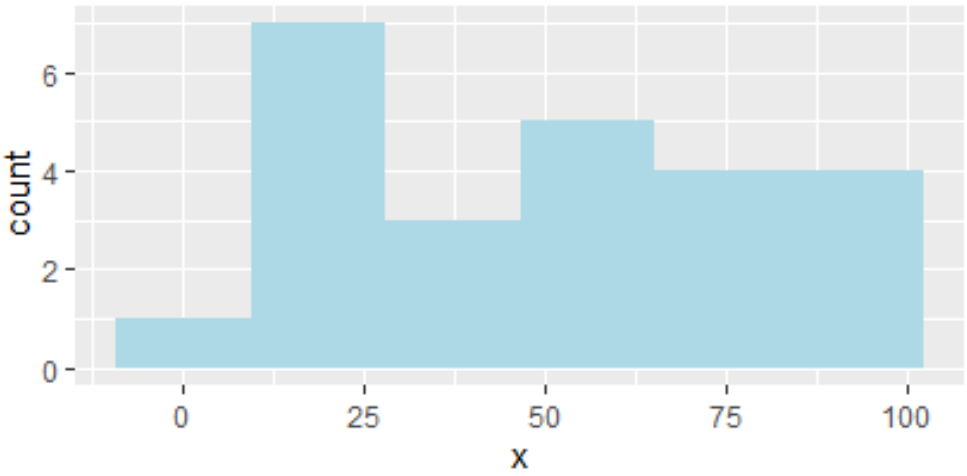


Figura 2.11: Histograma construído em R.

- **Gráfico de linhas:** utilizado quando estamos interessados em expressar variáveis que representam passagem de tempo, ou seja, mostrar a evolução histórica.

Um gráfico de linhas pode ser criado em Python simplesmente chamando a função `plot()` e passando como parâmetros os valores de x e y. O exemplo a seguir apresenta apenas cinco pares de valores (x,y). Digite o trecho de código a seguir e veja o resultado:

```
# Imports de bibliotecas
from matplotlib import pyplot
# Valores de x
x = [3, 10, 11, 13, 18, 20, 22, 24, 32, 43, 45, 51, 56, 63, 64, 67, 69,
77, 78, 81, 83, 88, 90, 96]
# Valores de y
y = [2,    2, 9, 1, 6, 6, 0, 6, 9, 1, 8, 4, 1, 9, 7, 3, 5, 4, 1, 6, 0, 2,
5, 7]
# Criação do gráfico
pyplot.plot(x, y)
# Exibição do gráfico
pyplot.show()
```

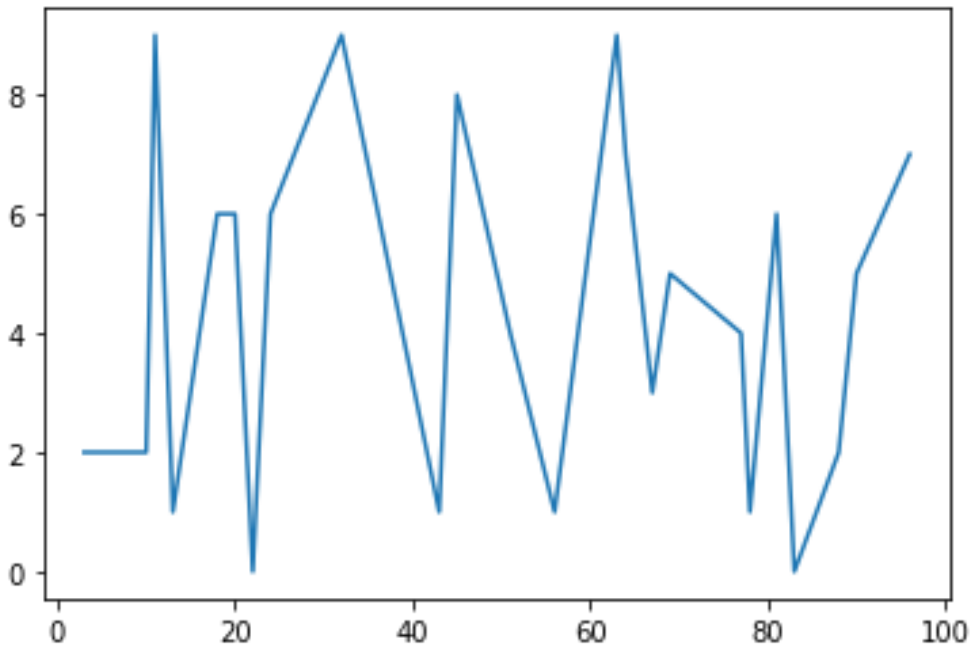


Figura 2.12: Gráfico de linhas construído em Python.

Um gráfico de linhas pode ser criado em R simplesmente chamando a função `ggplot()` e passando como parâmetros os dados, eixos x, eixo y e a geometria de interesse: **geom_line()**. Digite o trecho de código a seguir e veja o resultado:

```
library(ggplot2)
# Valores de x
x = c(3, 10, 11, 13, 18, 20, 22, 24, 32, 43, 45, 51, 56, 63, 64, 67, 69,
77, 78, 81, 83, 88, 90, 96)
# Valores de y
y = c(2, 2, 9, 1, 6, 6, 0, 6, 9, 1, 8, 4, 1, 9, 7, 3, 5, 4, 1, 6, 0, 2, 5,
7)
# Criando o data.frame
dados=data.frame(x,y)
# Criação do Gráfico e Exibição
ggplot(dados, aes(x, y)) +
  geom_line(col="blue")
```

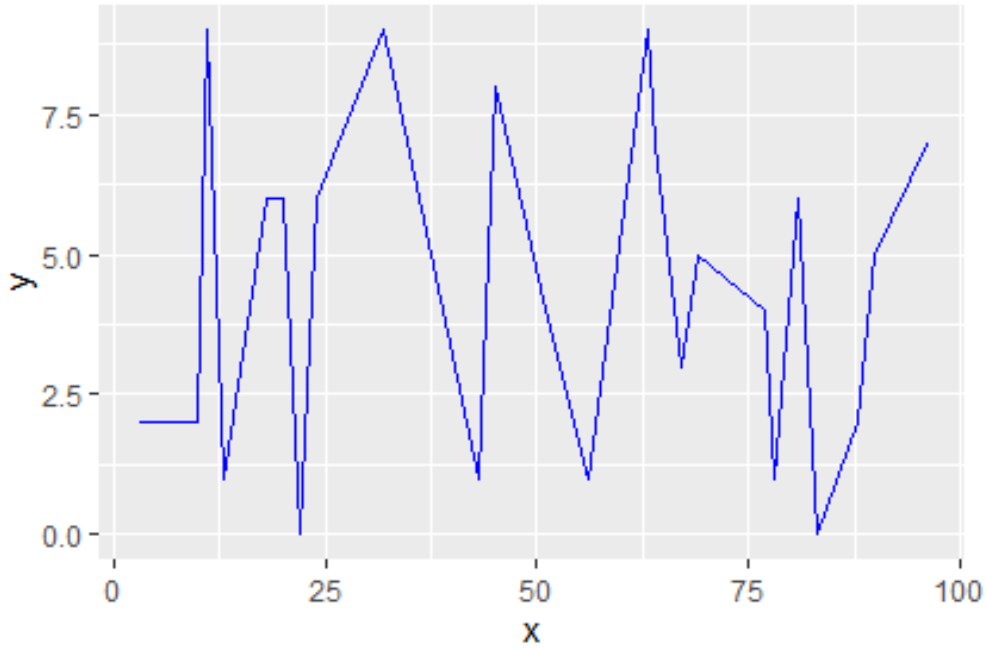


Figura 2.13: Gráfico de linhas construído em R.

2.4 Correlação

A **correlação** é uma medida estatística que descreve o grau de relacionamento linear entre duas variáveis. Ela indica se, e como, as duas variáveis se movem juntas. Quando duas variáveis estão correlacionadas, o comportamento de uma delas está relacionado ao comportamento da outra, de alguma forma. A correlação é usada em Ciência de Dados principalmente para:

- Entender relações entre variáveis: ela permite aos cientistas de dados identificar se existe uma relação entre duas variáveis e como essa relação se manifesta. Por exemplo, em um conjunto de dados de vendas de um produto, a correlação pode ajudar a determinar se a publicidade gasta em marketing afeta as vendas do produto.
- Seleção de características: em tarefas de modelagem preditiva, a correlação pode ser usada para selecionar as variáveis mais

relevantes. Variáveis altamente correlacionadas podem fornecer informações semelhantes, o que pode levar à redundância nos modelos. Portanto, pode ser desejável manter apenas uma das variáveis altamente correlacionadas.

- Validação de modelos: a correlação pode ser usada para avaliar o desempenho de modelos preditivos. Comparar a correlação entre as previsões do modelo e os valores reais ajuda a determinar o quão bem o modelo está se ajustando aos dados.
- Detecção de multicolinearidade: em regressões múltiplas, a correlação entre as variáveis independentes é usada para detectar multicolinearidade. Isso ocorre quando duas ou mais variáveis independentes estão altamente correlacionadas, o que pode prejudicar a precisão das estimativas do modelo.
- Análise exploratória de dados: a correlação é uma ferramenta útil na análise exploratória de dados para identificar relações potenciais que podem orientar análises subsequentes.

Existem várias formas de calcular a correlação, sendo a Correlação de Pearson e a Correlação de Spearman as mais comuns:

- **Correlação de Pearson:** mede a relação linear entre duas variáveis contínuas. Varia de -1 (correlação negativa perfeita) a 1 (correlação positiva perfeita), com 0 indicando nenhuma correlação linear.
- **Correlação de Spearman:** também conhecida como Correlação de Classificação, avalia a relação monotônica entre duas variáveis, independentemente de ser linear. É útil quando as variáveis não têm distribuições normais. Este método é baseado na classificação dos valores coletados e não nos próprios valores, podendo ser usado para variáveis em escalas ordinais.

É importante notar que correlação não implica causalidade. Ou seja, apenas porque duas variáveis estão correlacionadas, não significa que uma causa a outra. Outros fatores podem estar envolvidos, e a causalidade geralmente requer evidências adicionais além da correlação. Em resumo, a correlação é uma ferramenta valiosa em

Ciência de Dados para entender relações entre variáveis, validar modelos e auxiliar na tomada de decisões com base em dados.

Vejamos um exemplo simples de cálculo da Correlação de Pearson em Python:

```python
import numpy as np

# Dados de exemplo (duas variáveis)
variavel1 = [7, 9, 4, 6, 1]
variavel2 = [2, 3, 4, 5, 6]

# Calculando a correlação de Pearson
correlacao = np.corrcoef(variavel1, variavel2)[0, 1]

print("Correlação de Pearson:", correlacao)
```

A saída resultante será:

```
Correlação de Pearson: -0.7777137710478189
```

O código equivalente em R seria o seguinte:

```r
# Dados de exemplo (duas variáveis)
variavel1 <- c(7, 9, 4, 6, 1)
variavel2 <- c(2, 3, 4, 5, 6)

# Calculando a correlação de Pearson
correlacao <- cor(variavel1, variavel2)

print(paste("Correlação de Pearson:", correlacao))
```

A saída resultante será:

```
[1] "Correlação de Pearson: -0.777713771047819"
```

Quanto mais próximo o valor da Correlação de Pearson estiver de 1, mais forte será a correlação positiva entre as variáveis, e quanto mais próximo de -1, mais forte será a correlação negativa. Um valor próximo de 0 indica uma correlação fraca ou ausência de correlação linear.

A correlação entre duas variáveis é representada por um número, e a correlação entre mais de duas variáveis é representada por meio de uma matriz de correlação, que é útil para entender as relações entre diferentes variáveis em um conjunto dados. Vejamos um exemplo de como calcular uma matriz de correlação em Python usando a biblioteca Pandas e Matplotlib para visualização:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Criando um DataFrame de exemplo com algumas variáveis
dados = {
    'Idade': [45, 20, 25, 40, 34],
    'Salário': [10000, 70000, 5000, 9000, 8000],
    'Experiência': [13, 2, 5, 10, 12]
}

df = pd.DataFrame(dados)

# Calculando a matriz de correlação
matriz_correlacao = df.corr()

# Plotando a matriz de correlação como um gráfico de calor com valores
plt.figure(figsize=(8, 6))
plt.imshow(matriz_correlacao, cmap='coolwarm', interpolation='nearest')
plt.colorbar()

# Adicionando os valores de correlação nas células do gráfico
for i in range(len(df.columns)):
    for j in range(len(df.columns)):
        plt.text(i, j, f'{matriz_correlacao.iloc[i, j]:.2f}', ha='center',
va='center', color='black', fontsize=12)

plt.xticks(np.arange(len(df.columns)), df.columns, rotation=45)
plt.yticks(np.arange(len(df.columns)), df.columns)
```

```
plt.title('Matriz de Correlação')
plt.show()
```

A saída resultante será:

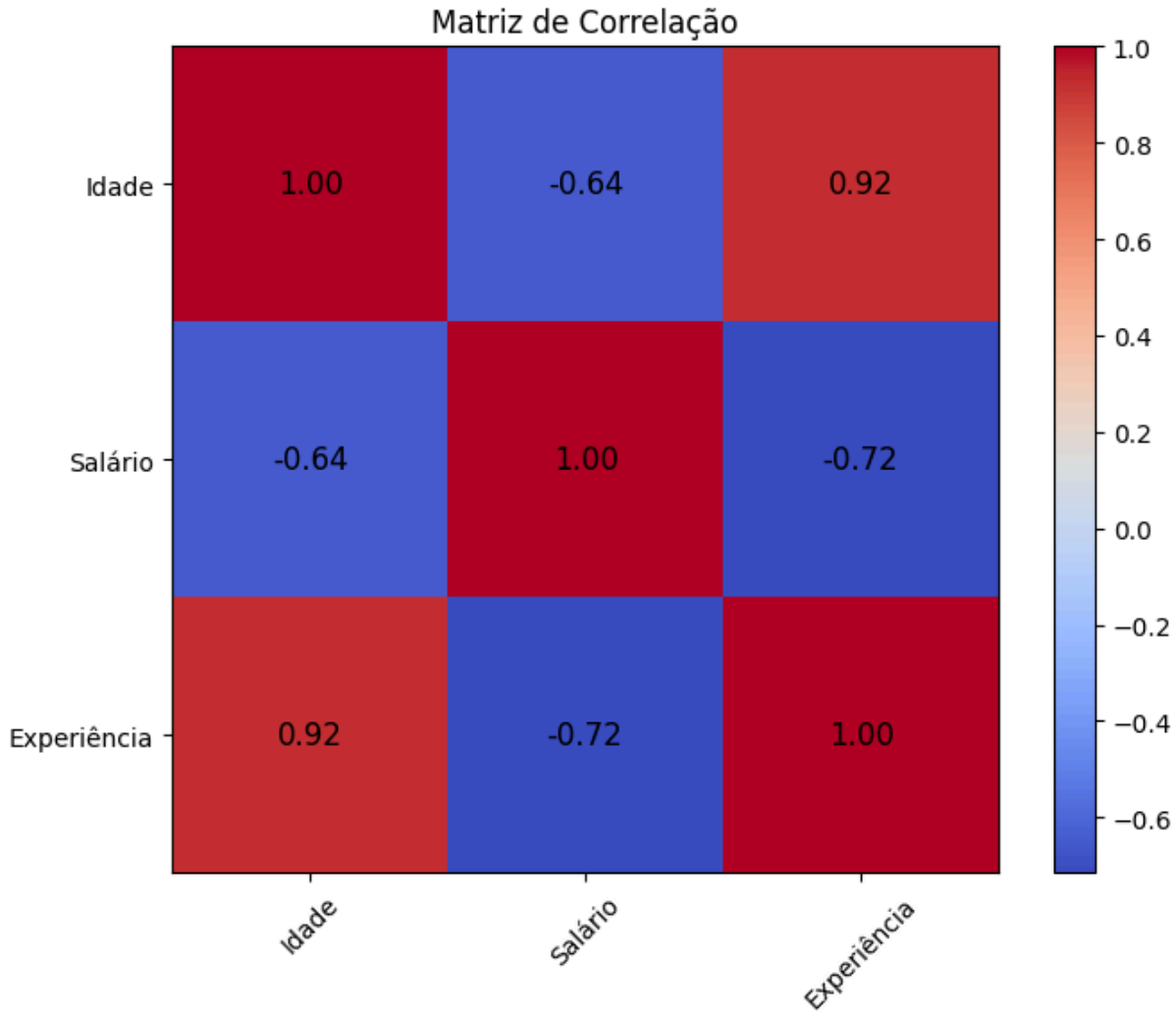


Figura 2.14: Matriz de Correlação de Pearson.

Para concluir este tópico, vejamos um exemplo de como calcular uma matriz de Correlação de Spearman em Python usando um conjunto de dados com variáveis ordinais e contínuas:

```
import pandas as pd
import scipy.stats as stats
```

```
# Criando um DataFrame de exemplo com variáveis ordinais e contínuas
dados = {
    'Satisfação': ['Baixa', 'Média', 'Alta', 'Média', 'Alta'],
    'Idade': [25, 30, 35, 40, 45],
    'Salário': [50000, 60000, 75000, 90000, 100000]
}

df = pd.DataFrame(dados)

# Mapeando as categorias para valores ordinais
mapeamento_ordem = {'Baixa': 1, 'Média': 2, 'Alta': 3}
df['Satisfação'] = df['Satisfação'].map(mapeamento_ordem)

# Calculando a matriz de correlação de Spearman
matriz_correlacao_spearman = df.corr(method='spearman')

# Imprimindo a matriz de correlação de Spearman
print("Matriz de Correlação de Spearman:")
print(matriz_correlacao_spearman)
```

A saída resultante será:

```
Matriz de Correlação de Spearman:
            Satisfação     Idade   Salário
Satisfação    1.000000  0.737865  0.737865
Idade         0.737865  1.000000  1.000000
Salário       0.737865  1.000000  1.000000
```

Este capítulo apresentou a primeira parte da Estatística Descritiva, abordando alguns conceitos básicos, tipos de variáveis, gráficos e correlação. O próximo capítulo apresentará a segunda parte da Estatística Descritiva, abordando medidas de tendência central e de dispersão.

CAPÍTULO 3
Estatística Descritiva: Medidas de tendência central e de dispersão

Conforme vimos no capítulo anterior, a Estatística Descritiva é o ramo da Estatística que se ocupa em organizar e descrever os dados, que podem ser expressos em tabelas e gráficos. No capítulo anterior, estudamos alguns conceitos básicos da Estatística Descritiva, tipos de variáveis e gráficos. Neste capítulo, abordaremos as subdivisões da Estatística Descritiva: medidas de tendência central (ou de posição) e medidas de dispersão.

3.1 Medidas de tendência central

As medidas de tendência central (ou de posição) possibilitam saber o grau de concentração dos dados. São uma forma de resumir os seus dados por meio de valores representativos do conjunto de dados. São exemplos: média, mediana, moda e medidas separatrizes.

Tipos de médias

A Média é considerada uma medida volátil, uma vez que é afetada por valores extremos e sensível a valores atípicos (*outliers*). Ao se somar ou subtrair uma constante a todas as observações do conjunto, a Média fica somada ou subtraída do valor da constante. Ao se efetuar a multiplicação ou divisão por uma constante a todas as observações, a Média fica multiplicada ou dividida pela mesma constante. Podemos utilizar diversos tipos de médias, conforme detalhado a seguir:

Média Aritmética (MA)

É a soma de todos os elementos do conjunto, dividida pelo número de elementos que compõem o conjunto. Sua fórmula é dada por:

$$\bar{x} = \frac{x_1 + x_2 + x_3 + \ldots + x_n}{n} = \frac{\sum_{i=1}^{n} x_i}{n}$$

Média Geométrica (MG)

É a raiz n-ésima do produto de todos os elementos que compõem o conjunto. *n* é o número de elementos que compõem o conjunto. Sua fórmula é dada por:

$$MG = \sqrt[n]{x_1 . x_2 . \ldots . x_n}$$

Média Harmônica (MH)

É o número de elementos, dividido pela soma dos inversos dos elementos que compõem o conjunto. Sua fórmula é dada por:

$$MH = \frac{n}{\left(\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}\right)}$$

Apesar de a Média ser fácil de computar, ela nem sempre é a melhor medida. Quando há uma alta variabilidade nos dados, medidas mais robustas (como a mediana e a moda) podem ser mais úteis.

As médias Geométricas e Harmônicas são bastante utilizadas pelos economistas ao calcular números índices para mensurar a inflação da economia de um país, deflacionar séries históricas de aluguéis e preço de imóveis, por meio de índices como IGP (Índice geral de preços) mantido pela FGV (Fundação Getúlio Vargas) e o IPCA (Índice de preços ao consumidor amplo) calculado mensalmente pelo IBGE para calcular a variação dos preços no comércio. A Média

Harmônica tem uma aplicação bastante importante em problemas de classificação de *Machine Learning*, sendo utilizada para calcular a métrica *F1 Score*, que é a Média Harmônica das métricas de precisão e *recall*.

Relação entre as médias: MA ≥ MG ≥ MH

Observação: o único caso em que MA = MG = MH é quando todos os elementos possuem o mesmo valor no conjunto de dados!

Média Ponderada (W)

Calculada por meio do somatório das multiplicações entre valores (x) e pesos divididos pelo somatório dos pesos (w). Sua fórmula é dada por:

$$W = \frac{\sum_{i=1}^{n} w_i X_i}{\sum_{i=1}^{n} w_i}$$

Média para dados agrupados em classes

Quando os dados estiverem agrupados em classes, usaremos a seguinte fórmula para Média Aritmética:

$$\bar{X} = \frac{\sum PM_i \,.\, f_i}{\sum fi} \text{ , onde } PM_i = \frac{l_{inf} + l_{sup}}{2}$$

Exemplo

Calcule a Média Aritmética, a Média Geométrica e a Média Harmônica para o seguinte conjunto de dados: {1,2,5,3,4}

Média Aritmética (MA):

$$MA = \bar{x} = \frac{x_1 + x_2 + x_3 + \ldots + x_n}{n} = \frac{\sum_{i=1}^{n} x_i}{n} \therefore \frac{(1+2+5+3+4)}{5} = 3{,}0$$

Podemos calcular a Média Aritmética com Python usando a função `mean()`, do pacote `statistics`. Para tal, execute o bloco de código a seguir:

```
# Import da função
from statistics import mean
# Dados
dados = [1,2,5,3,4]
# Calculando a média aritmética
mean(dados)
```

A saída será:

```
3
```

Para calcular no R a Média Aritmética por meio da função `mean()`, basta digitar:

```
# Dados
dados = c(1,2,5,3,4)
# Calculando a média aritmética
mean(dados)
`
```

A saída resultante será:

```
[1] 3
```

Verificamos que a Média Aritmética do conjunto de dados {1,2,5,3,4} é 3.

Média Geométrica (MG):

$$MG = \sqrt[n]{x_1 . x_2 . \ldots . x_n} \therefore \sqrt[5]{1.2.5.3.4} \approx 2{,}605$$

Podemos calcular a Média Geométrica com Python usando a função `gmean()`, do pacote `scipy.stats.mstats`. Para tal, execute o bloco de código a seguir:

```
# Import da função
from scipy.stats.mstats import gmean
# Dados
dados = [1,2,5,3,4]
# Calculando a média geométrica
gmean(dados)
```

A saída resultante será:

```
2.6051710846973517
```

Para calcular no R a Média Geométrica manualmente, basta digitar:

```
# Dados
dados = c(1,2,5,3,4)
# Calculando a média geométrica
exp(mean(log(dados)))
```

A saída resultante será:

```
[1] 2.605171
```

Outra opção no R é usar a função `geoMean()`, da biblioteca **EnvStats**, que tem como saída um resultado similar:

```
# Instalando a biblioteca
install.packages("EnvStats")
# Carregando a biblioteca
library(EnvStats)
# Calculando a média geométrica
geoMean(dados)
```

Média Harmônica (MH):

$$MH = \frac{n}{\left(\frac{1}{x_1}+\frac{1}{x_2}+\cdots+\frac{1}{x_n}\right)} \therefore \frac{5}{\left(\frac{1}{1}+\frac{1}{2}+\frac{1}{5}+\frac{1}{3}+\frac{1}{4}\right)} \approx 2{,}19$$

Podemos calcular a Média Harmônica com Python usando a função `harmonic_mean()`, do pacote `statistics`. Para tal, execute o bloco de código a seguir:

```
# Import da função
from statistics import harmonic_mean
# Dados
dados = [1,2,5,3,4]
# Calculando a média harmônica
harmonic_mean(dados)
```

A saída resultante será:

```
2.18978102189781
```

Para calcular no R a Média Harmônica manualmente, basta digitar:

```
# Dados
dados= c (1,2,5,3,4)
# Calculando a média harmônica
1/mean(1/dados)
```

A saída será:

```
[1] 2.189781
```

Outra opção no R é usar a função `harmonic.mean()` da biblioteca **psych**, que tem como saída um resultado similar:

```
# Instalando a biblioteca
install.packages("psych")
# Carregando a biblioteca
library(psych)
# Calculando a média Harmônica
harmonic.mean(dados)
```

Como já sabíamos: MA > MG > MH.

Mediana

É o valor da variável que divide os dados ordenados em duas partes de igual frequência.

A Mediana é considerada uma medida robusta, uma vez que é menos afetada por valores extremos. Ao se somar ou subtrair uma constante a todas as observações do conjunto, a Mediana fica somada ou subtraída do valor da constante. Ao se efetuar a multiplicação ou divisão por uma constante a todas as observações, a Mediana fica multiplicada ou dividida pela mesma constante.

Para calcular a Mediana em dados não agrupados em classes, deve-se utilizar o seguinte procedimento:

1. Colocar os dados em rol, ou seja, ordenar os dados de forma crescente ou decrescente.
2. Observar a paridade do *n*, pois o cálculo da Mediana difere para *n* par e *n* ímpar.
3. Se *n* é ímpar, temos uma posição central única, dada por PC (Posição Central)= *(n+1)/2*. Após calcularmos PC, a Mediana será o valor que ocupa a posição central.
4. Se *n* é par, calcularemos duas posições centrais, PC1 (Posição Central 1)= *n/2* e PC2 (Posição Central 2) = *(n/2) + 1*. Após calcularmos PC1 e PC2, a Mediana será a Média Aritmética de PC1 e PC2.

Exemplo

Calcule a Mediana das observações: {7,1,5,2,3,1,6}.

1. O primeiro passo é ordenar os dados do enunciado. Após fazer a ordenação, teremos: {1,1,2,3,5,6,7}.
2. Em seguida, precisamos calcular o PC. Como n=7, ou seja, temos 7 elementos, e este número é ímpar, temos somente um

PC, que pode ser calculado por: PC = (n+1)/2 ∴ PC = (7+1)/2 = 4

Concluímos que a Mediana é a observação cuja posição, com os dados ordenados (em rol), é a quarta. Logo, Md = 3.

Podemos calcular a Mediana com Python usando a função `median()`, do pacote `statistics`. Para tal, execute o bloco de código a seguir:

```
# Import da função
from statistics import median
# Dados
dados = [7,1,5,2,3,1,6]
# Calculando a mediana
median(dados)
```

Saída:

```
3
```

Para calcular a Mediana das observações no R por meio da função `median()´:

```
# Dados
dados = c(7,1,5,2,3,1,6)
# Calculando a mediana
median(dados)
```

Saída:

```
[1] 3
```

Exemplo

Calcule a Mediana das observações: {1,2,1,1,4,5,3,6}.

1. O primeiro passo é ordenar os dados do enunciado. Após fazer a ordenação, teremos: {1,1,1,2,3,4,5,6}.
2. Em seguida, precisamos calcular o PC. Como n=8, ou seja, temos 8 elementos, e este número é par, temos dois PCs, que

podem ser calculados por: PC1= n/2 = 8/2 = 4 e PC2= (n/2)+1 = 5.

Logo, a nossa Mediana será a Média Aritmética da quarta observação e a quinta observação: Md = (2+3)/2 = 2,5.

No Python:

```
# Import da função
from statistics import median
# Dados
dados = [1,2,1,1,4,5,3,6]
# Calculando a mediana
median(dados)
```

Saída:

```
2.5
```

No R:

```
# Dados
dados = c(1,2,1,1,4,5,3,6)
# Calculando a mediana
median(dados)
```

Saída:

```
[1] 2.5
```

Moda

É o valor que possui a maior frequência simples no conjunto de dados, consequentemente, o de maior probabilidade de ocorrência em um conjunto de dados não agrupados em classes.

A Moda é considerada uma medida robusta, uma vez que não é afetada por valores extremos. Ao se somar ou subtrair uma constante a todas as observações do conjunto, a Moda fica somada ou subtraída do valor da constante. Ao se efetuar a multiplicação ou

divisão por uma constante a todas as observações, a Moda fica multiplicada ou dividida pela mesma constante.

Para calcular a Moda em dados não agrupados em classes, basta contar o número de ocorrências de cada valor. A Moda será o valor com o maior número de ocorrências. Vale observar que, em um conjunto de dados, a Moda pode ser única (**unimodal**), ter dois valores (**bimodal**), ou mesmo não existir (**amodal**), como veremos nos exemplos a seguir.

Exemplo

Calcule a Moda do conjunto {4,5,4,6,5,8,4}.

Vamos construir a tabela de frequências para facilitar a nossa visualização:

x	**f_i**
4	3
5	2
6	1
8	1

Percebemos pela tabela de frequências que a Moda é 4, pois possui a maior frequência simples do conjunto, logo a Moda é única e chamada de **unimodal**.

Podemos calcular a Moda com Python usando a função `mode()`, do pacote `statistics`. Para tal, execute o bloco de código a seguir:

```
# Import da função
from statistics import mode
# Dados
dados = [4,5,4,6,5,8,4]
# Calculando a moda
mode(dados)
```

A saída resultante será:

```
4
```

Para calcular a Moda no R vamos utilizar a função `Mode()` da biblioteca **DescTools**, seu output não só dá o valor mais frequente (Moda) como também a sua respectiva frequência e, em caso de amodal, retornará (NA).

```
# Dados
dados = c(4,5,4,6,5,8,4)
# Instalando a biblioteca
install.packages("DescTools")
# Carregando a biblioteca
library(DescTools)
# Calculando a moda
Mode(dados)
```

A saída resultante será:

```
[1] 4
attr(,"freq")
[1] 3
```

Vejamos agora um exemplo de quando temos mais de uma Moda em um conjunto de dados.

Exemplo

Calcule a moda do conjunto {4,5,4,6,5,8,4,4,5,5}.

Vamos construir a tabela de frequências para facilitar a nossa visualização:

x	**f_i**
4	4
5	4
6	1

x	f_i
8	1

Percebemos pela tabela de frequências que a Moda possui dois valores, pois duas observações do conjunto se repetem 4 vezes. Portanto, a moda é 4 e 5 e é chamada de **bimodal**. Vejamos como fica o código em Python e em R a seguir. Note que em Python utilizaremos a função `multimode()` do pacote `statistics`, disponível a partir da versão 3.8 do Python.

```
# Import da função
from statistics import multimode
# Dados
dados = [4,5,4,6,5,8,4,4,5,5]
# Calculando a moda
multimode(dados)
```

Saída:

```
[4, 5]
```

```
# Dados
dados = c(4,5,4,6,5,8,4,4,5,5)
# Instalando a biblioteca
install.packages("DescTools")
# Carregando a biblioteca
library(DescTools)
# Calculando a moda
Mode(dados)
```

Saída:

```
[1] 4 5
attr(,"freq")
[1] 4
```

Finalmente, veremos agora um exemplo de quando a Moda não existe.

Exemplo

Calcule a Moda do conjunto {1,2,3,4,5}.

Vamos construir a tabela de frequências para facilitar a nossa visualização:

x	**f_i**
1	1
2	1
3	1
4	1
5	1

Percebemos pela tabela de frequências que a Moda não possui valor, pois nenhuma observação do conjunto se repete mais de uma vez, portanto, é chamada de **amodal**. Vejamos como fica o código em Python e em R a seguir. Note que em Python utilizaremos a função `multimode()` do pacote `statistics`, disponível a partir da versão 3.8 do Python.

```
# Import da função
from statistics import multimode
# Dados
dados = [1,2,3,4,5]
# Calculando a moda
multimode(dados)
```

Saída:

```
[1, 2, 3, 4, 5]
```

```
# Dados
dados = c(1,2,3,4,5)
# Instalando a biblioteca
install.packages("DescTools")
```

```
# Carregando a biblioteca
library(DescTools)
# Calculando a moda
Mode(dados)
```

Saída:

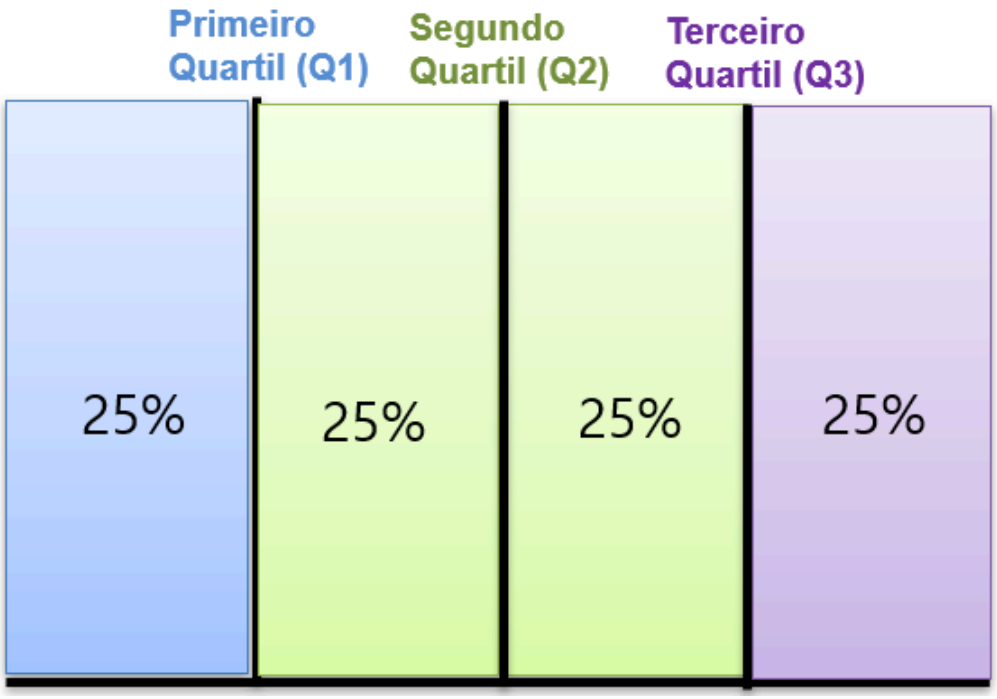
```
[1] NA
attr(,"freq")
[1] 1
```

Medidas separatrizes

As medidas separatrizes têm como objetivo dividir o conjunto de dados em *n* partes iguais. As mais utilizadas são os **Quartis** e os **Percentis**.

Quartis: dividem o conjunto em quatro partes iguais.

- Primeiro Quartil (Q1): é o valor que deixa o conjunto de dados 25% abaixo dele e 75% para cima.
- Segundo Quartil (Q2 = Mediana): é o valor que deixa o conjunto de dados 50% abaixo dele e 50% para cima, em duas partes de igual frequência.
- Terceiro Quartil (Q3): é o valor que deixa o conjunto de dados 75% abaixo dele e 25% para cima.



Percentis: dividem o conjunto em 100 partes iguais.

- Percentil Vinte e Cinco (P25 = Q1).
- Percentil Cinquenta (P50 = Q2 = Md).
- Percentil Setenta e Cinco (P75 = Q3).

Exemplo

Calcule a Mediana, Primeiro Quartil e Terceiro Quartil das observações: {3,1,5,2,3,1,7}.

Primeiramente, vamos ao cálculo da Mediana:

1. O primeiro passo é ordenar os dados. Após fazer a ordenação, teremos: {1,1,2,3,3,5,7}.
2. Em seguida, precisamos calcular o PC. Como n=7, ou seja, temos 7 elementos, e este número é ímpar, temos somente um PC, que pode ser calculado por: PC = (n+1)/2 = 4

Portanto, a Mediana será a observação cuja posição é a quarta no nosso conjunto de dados e será Md=3.

Finalmente, vamos ao cálculo dos Quartis:

- O Primeiro Quartil da nossa distribuição é aquele que deixa o conjunto dividido em 25% para baixo e 75% para cima, portanto este valor é a Mediana da "Primeira parte", logo Q1 é 1.
- O Terceiro Quartil da nossa distribuição é aquele que deixa o conjunto dividido em 75% para baixo e 25% para cima, portanto este valor é a Mediana da "segunda parte", logo Q3 é 5.

Para calcular os Quantis (Percentis, Quartis, Decis...) no Python, utilizaremos a função `quantile()`, do pacote `numpy`, passando os dados como primeiro parâmetro e a probabilidade equivalente ao Quantil desejado como segundo parâmetro:

```
# Import da função
from statistics import median
import numpy as np
# Dados
dados = [3,1,5,2,3,1,7]
# Mediana
print(median(dados))
print(np.quantile(dados, 0.25))
print(np.quantile(dados, 0.75))
```

Saída:

```
3
1.5
4.0
```

Já para calcular os Quantis (Percentis, Quartis, Decis...) no R, utilizaremos a função `quantile()`, passando os dados como primeiro parâmetro e a probabilidade equivalente ao Quantil desejado como segundo parâmetro. Também podemos mostrar de uma vez o Primeiro Quartil (Q1), Segundo Quartil (Q2) e Terceiro Quartil (Q3) usando a função `summary()`:

```
# Dados
dados = c(3,1,5,2,3,1,7)
```

```
# Mediana
print(median(dados))
# Primeiro Quartil
print(quantile(dados,.25))
# Terceiro Quartil
print(quantile(dados,.75))
# Achando o Primeiro Quartil (Q1), Segundo Quartil (Q2) e Terceiro Quartil
(Q3) de uma vez.
print(summary(dados))
```

Saída:

```
[1] 3
25%
1.5
75%
  4
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.500   3.000   3.143   4.000   7.000
```

Medidas de assimetria

As medidas de assimetria possibilitam analisar uma distribuição em relação a sua Moda, Mediana e Média. Tentam mensurar como e quanto as distribuições se afastam da simetria da curva da distribuição normal de probabilidade.

Pense no conceito de simetria fazendo uma analogia a um espelho. Se traçarmos um eixo vertical no meio da curva dos dados e enxergarmos o mesmo de um lado e de outro, significa que seus dados são simétricos. Neste caso, a Média, a Mediana e a Moda serão iguais, o que não será verdade no caso de dados assimétricos. Esse conceito é ilustrado pela figura a seguir:

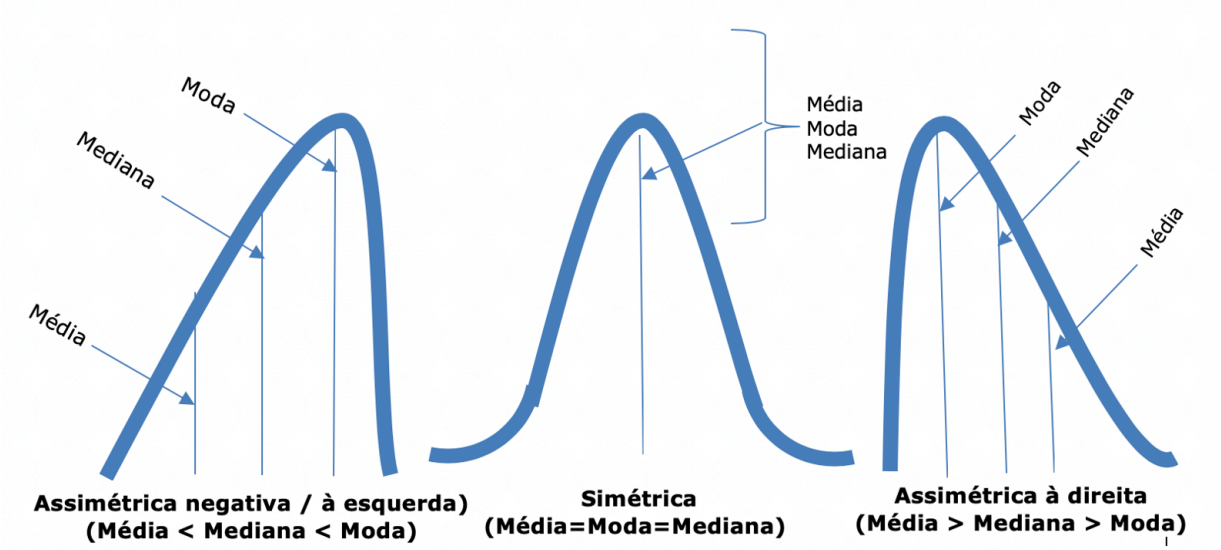


Figura 3.10: Distribuições Simétricas e Assimétricas.

Para calcular a assimetria no R utilize a função `skewness()` da biblioteca **e1071**.

Coeficiente de assimetria de Pearson

Existem alguns coeficientes de assimetria de Pearson, mas falaremos somente do coeficiente de assimetria de Pearson baseado nos momentos, que tem a seguinte fórmula:

$$AS = \frac{\sum_i^N (X_i - \bar{X})^3}{(N-1) \times \sigma^3}$$

onde:

- AS = assimetria
- N = número de variáveis na distribuição
- X_i = variável aleatória
- $\bar{X}$ = média da distribuição
- σ = desvio padrão da distribuição

Quando *AS = 0*, a distribuição será simétrica, ou seja, terá ausência de assimetria. Ela terá o formato de uma distribuição normal de Probabilidade. Quando *AS > 0*, a distribuição será assimétrica à direita ou positiva, ou seja, terá concentração dos valores no gráfico à esquerda e a curva se alongará mais à direita do gráfico. Quando *AS < 0*, a distribuição será assimétrica à esquerda ou negativa, ou seja, terá concentração dos valores no gráfico à direita e a curva se alongará mais à esquerda do gráfico.

Vejamos como ficam os códigos em R e em Python para o cálculo do Coeficiente de Assimetria de Pearson:

```
# Instalando a biblioteca
#install.packages("e1071")
# Carregando a biblioteca
#library(e1071)
# Dados
x = c(8.0, 1, 2.5, 4, 28.0)
# Calculando a assimetria manualmente
n = length(x)
mean_ = sum(x) / n
a = (sum((x - mean_)**3) * n / ((n - 1) * (n - 2) * sd(x)**3))
# Calculando a assimetria usando a biblioteca
skewness(x,type = 2)
# AS > 0 (1.947043) --> distribuição será assimétrica à direita ou
positiva, terá concentração dos valores no gráfico à esquerda e a curva se
alongará mais à direita do gráfico)
```

A saída resultante será:

```
R[write to console]:
Attaching package: ‘e1071’

R[write to console]: The following objects are masked from
‘package:EnvStats’:

    kurtosis, skewness

[1] 1.947043
```

```
# Import da função
from scipy.stats import skew
# Dados
x = [8.0, 1, 2.5, 4, 28.0]
# Calculando a assimetria manualmente
n = len(x)
mean_ = sum(x) / n
var_ = sum((item - mean_)**2 for item in x) / (n - 1)
std_ = var_ ** 0.5
skew_ = (sum((item - mean_)**3 for item in x) * n / ((n - 1) * (n - 2) *
std_**3))
# Calculando a assimetria usando a biblioteca
skew_
# AS > 0 (1.9470432273905929) --> distribuição será assimétrica à direita
ou positiva, terá concentração dos valores no gráfico à esquerda e a curva
se alongará mais à direita do gráfico)
```

A saída resultante será:

```
1.9470432273905929
```

3.2 Medidas de dispersão

As medidas de dispersão, também chamadas de Medidas de Variação, nos permitem saber o grau de dispersão dos dados em relação a uma medida de tendência central (geralmente a Média). Medem se os dados estão compactados ou espalhados. São exemplos de medidas de dispersão: Amplitude, Variância, Desvio Padrão, Coeficiente de Variação. Vamos entender cada conceito associado a população e amostra e, em seguida, veremos um exemplo.

População

Amplitude Populacional: é a diferença entre o maior e o menor valor da população:

H = (máximo - mínimo)

Variância Populacional: é o valor médio dos quadrados dos desvios de cada valor da população em relação ao valor médio da população:

$$\sigma^2 = \frac{(x_1-\bar{x})^2 + (x_2-\bar{x})^2 + \cdots + (x_N-\bar{x})^2}{N} = \frac{\sum_{i=1}^{N}(x_i-\bar{x})^2}{N}$$

A Variância é considerada uma medida de dispersão absoluta. Ao se somar ou subtrair uma constante a todas as observações do conjunto, a Variância fica inalterada. Ao se efetuar a multiplicação ou divisão por uma constante a todas as observações, a Variância fica multiplicada ou dividida pela mesma constante ao quadrado.

Desvio Padrão Populacional: é a raiz quadrada da Variância Populacional:

$$\sigma = \sqrt{\sigma^2}$$

O Desvio Padrão é considerado uma medida de dispersão absoluta. Ao se somar ou subtrair uma constante a todas as observações do conjunto, o Desvio Padrão fica inalterado. Ao se efetuar a multiplicação ou divisão por uma constante a todas as observações, o Desvio Padrão fica multiplicado ou dividido pela mesma constante. É mais fácil de interpretar do que a Variância por estar na mesma escala que os dados originais.

Coeficiente de Variação Populacional: é a razão entre o Desvio Padrão Populacional e a Média Populacional, e é um indicador de homogeneidade dos dados da população:

$$CV_x = \frac{\sigma}{\mu'},$$

O Coeficiente de Variação é considerado uma medida de dispersão relativa e é a melhor de todas as medidas de dispersão, uma vez que leva em consideração não somente a variabilidade como a média do conjunto de dados. Ao se somar ou subtrair uma constante a todas as observações do conjunto, o denominador do Coeficiente de Variação fica somado ou subtraído do valor da constante. Ao se efetuar a multiplicação ou divisão por uma constante a todas as observações, o Coeficiente de Variação fica inalterado.

Observação: o Coeficiente de Variação é a única medida de variação adimensional (não possui unidade de medida). Em geral, consideramos um Coeficiente de Variação < 25%, um bom indicador de homogeneidade dos dados.

Amostra

Amplitude Amostral: é a diferença entre o maior e o menor valor da amostra: *h = (máximo – mínimo)*

Variância Amostral: é o valor médio dos quadrados dos desvios de cada valor da amostra em relação ao valor médio da amostra:

$$s^2 = \frac{(x_1-\bar{x})^2 + (x_2-\bar{x})^2 + \cdots + (x_n-\bar{x})^2}{n-1} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1}$$

Note que dividimos a Variância Amostral por (n-1), diferentemente da Variância Populacional, que dividimos por (N).

Desvio Padrão Amostral: é a raiz quadrada da Variância Amostral:

$$s = \sqrt{s^2}$$

Coeficiente de Variação Amostral: é a razão entre o Desvio Padrão Amostral e a Média Amostral, e é um indicador de homogeneidade dos dados da amostra:

$$CV_x = \frac{s}{\bar{x}}$$

Exemplo

Considere a amostra: {3,4,5,6,12}. Calcule: Amplitude, Variância, Desvio padrão e Coeficiente de Variação da amostra.

Amplitude: *h = (máximo - mínimo) = 12 – 3 = 9*

Variância: $s^2 = ((3-6)^2+(4-6)^2+(5-6)^2+(6-6)^2+(12-6)^2)/(5-1)= 50/4 = 12,5$

Desvio Padrão: *s = $|sqrt{12,5}$ = 3,535534.*

Coeficiente de Variação $cv_x = 3,535534/6 = 0,5892557$ *ou 58,92557%*

Como podemos notar, o conjunto de dados é considerado heterogêneo, uma vez que o CV (coeficiente de variação) foi muito maior que o limite estabelecido de 25%.

Utilizando o Python:

```
# Imports das bibliotecas
from statistics import variance
from statistics import stdev
from statistics import mean
import numpy as np
# Dados
dados = [3,4,5,6,12]
```

```
# Amplitude
print(np.max(dados)-np.min(dados))
# Variância
print(variance(dados))
# Desvio padrão
print(stdev(dados))
# Coeficiente de variação
print(stdev(dados)/mean(dados)*100)
```

Saída:

```
9
12.5
3.5355339059327378
58.92556509887896
```

Utilizando o R:

```
# Dados
dados = c(3,4,5,6,12)
# Amplitude
print(diff(range(dados)))
# Variância
print(var(dados))
# Desvio padrão
print(sd(dados))
# Coeficiente de variação
print(sd(dados)/mean(dados)*100)
```

Saída:

```
[1] 9
[1] 12.5
[1] 3.535534
[1] 58.92557
```

3.3 Boxplot, ou diagrama de caixas

Para trabalharmos com as medidas de tendência central e de dispersão graficamente, podemos usar o diagrama de caixas (em inglês, *boxplot*), que é uma representação gráfica da distribuição dos dados. Esse diagrama nos dá informação da assimetria da distribuição, da presença de *outliers* (valores atípicos) e da variabilidade dos dados por meio da amplitude (Máx-Min). Esse conceito é ilustrado pela figura a seguir:

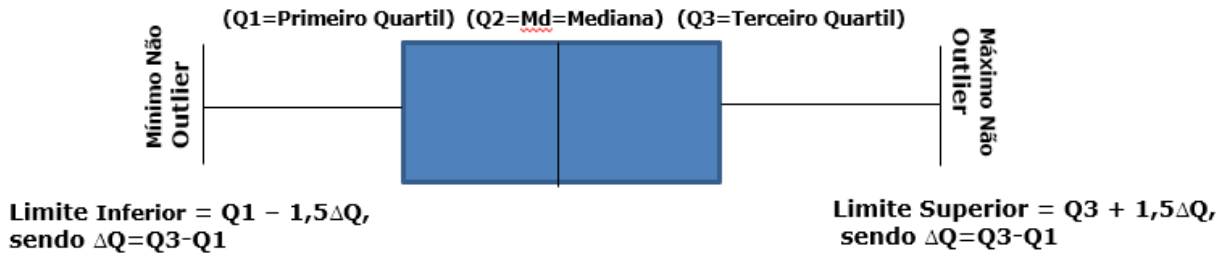


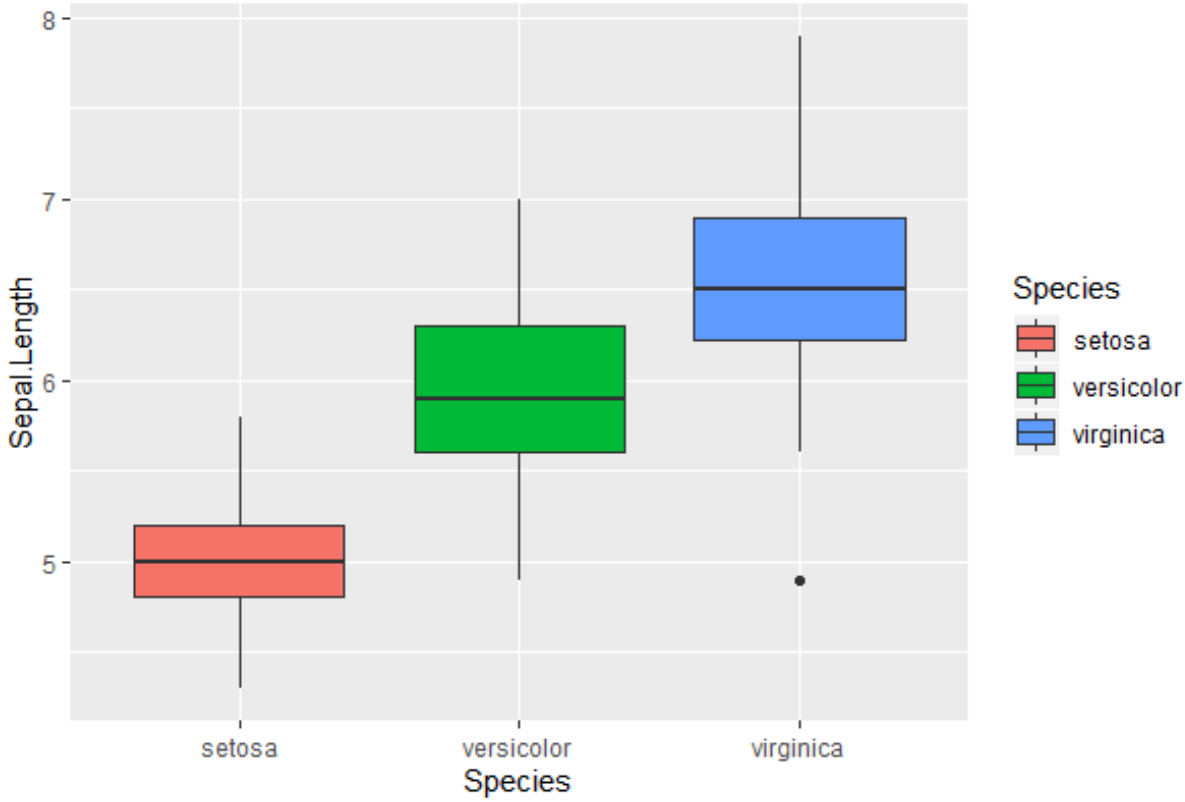
Figura 3.18: Diagrama de Boxplot.

Há diversas bibliotecas em Python que podemos utilizar para gerar boxplots. No exemplo a seguir, utilizaremos as bibliotecas **Seaborn** e **Pyplot** para gerar boxplots agrupados por espécies da característica "Petal Length" do conhecido dataset de flores Iris (para saber mais sobre este dataset https://en.wikipedia.org/wiki/Iris_flower_data_set):

```
# Imports das bibliotecas
import seaborn as sns
import matplotlib.pyplot as plt

# Carregando o dataset
dados = sns.load_dataset('iris')
# Plotando um boxplot por species da característica petal length
sns.boxplot(x=dados["species"], y=dados["petal_length"])
plt.show()
```

Um boxplot pode ser criado em R chamando a função `ggplot()` e passando como parâmetros os dados, os eixos x e y e a geometria de interesse: `geom_boxplot()` :

```r
# Carregando a biblioteca
library(ggplot2)
# Utilizando o dataset nativo Iris
dados = iris
# Plotando um boxplot por Species da característica Petal.Length
ggplot(dados, aes(x=Species, y=Petal.Length, fill=Species)) +
  geom_boxplot()
```



3.4 Resumo

A figura a seguir resume os ramos da Estatística Descritiva.

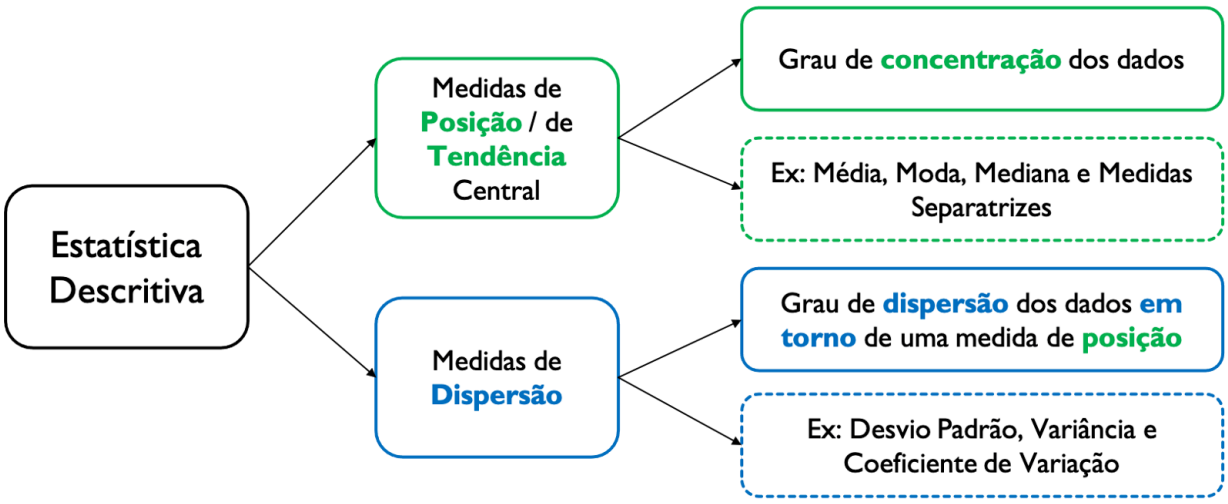


Figura 3.20: Ramos da Estatística Descritiva.

As tabelas a seguir resumem as medidas de posição e dispersão apresentadas neste capítulo:

Medida	Tipo de Medida	Sensível a Valores Extremos?
Média	Posição	Sim
Moda	Posição	Não
Mediana	Posição	Não
Variância	Dispersão	-
Desvio Padrão	Dispersão	-
Coeficiente de Variação	Dispersão	-

Medida	Somar uma constante (K) a todos os elementos	Subtrair uma constante (K) a todos os elementos	Multiplicar uma constante (K) a todos os elementos	Div un const (K) todo eleme
Média	Somar K	Subtrair K	Multiplicar por K	Dividir K
Moda	Somar K	Subtrair K	Multiplicar por K	Dividir K
Mediana	Somar K	Subtrair k	Multiplicar por K	Dividir K
Variância	Não se altera	Não se altera	Multiplicar por k^2	Dividir k^2
Desvio Padrão	Não se altera	Não se altera	Multiplicar por K	Dividir K
Coeficiente de Variação	Somar K ao denominador	Subtrair K ao denominador	Não se altera	Não se altera

Este capítulo apresentou a segunda parte da Estatística Descritiva, abordando medidas de tendência central e de dispersão. O próximo capítulo apresentará a primeira parte do conteúdo referente a cálculo de probabilidades.

CAPÍTULO 4
Cálculo das Probabilidades: Conceitos e fundamentos

Conforme vimos nos capítulos anteriores, a estatística pode ser subdivida em três grandes áreas: Estatística Descritiva, Probabilidade e Inferência Estatística. Neste capítulo, serão abordados assuntos relacionados à Probabilidade, que nos permite descrever os fenômenos aleatórios, ou seja, aqueles em que está presente a incerteza. Serão apresentados conceitos fundamentais em Probabilidade, como axiomas de Kolmogorov, probabilidade da união e intersecção, leis de Morgan, lei da probabilidade total, probabilidade condicional e teorema de Bayes.

4.1 Conceitos fundamentais em probabilidade

A teoria das probabilidades nos permite descrever os fenômenos aleatórios, ou seja, aqueles em que a incerteza se encontra presente. Assim, existem três conceitos fundamentais em probabilidade: experimento aleatório, espaço amostral e evento, ilustrados pela figura a seguir:

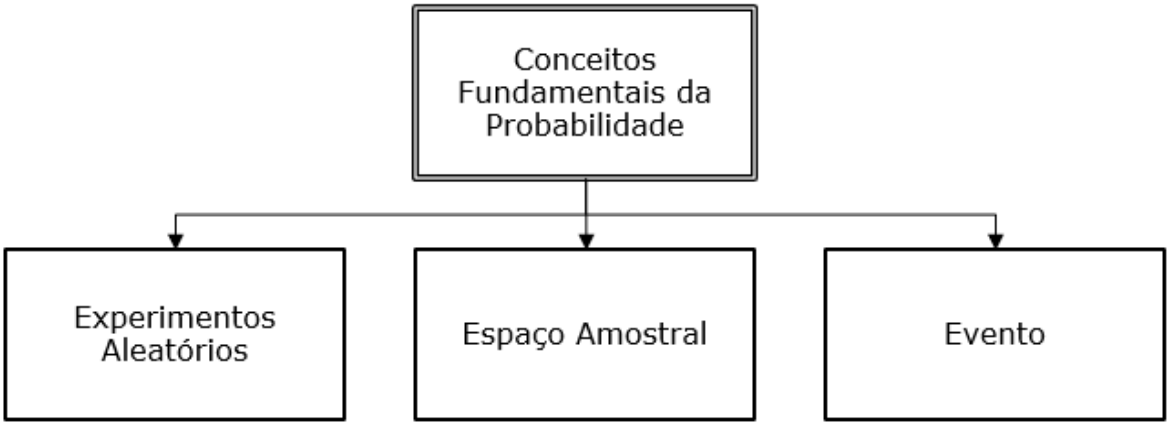


Figura 4.1: Conceitos fundamentais em probabilidade.

Experimento aleatório

É uma ação cujo resultado não pode ser previsto. Embora não se possa determinar seu resultado, é possível mapear o conjunto de todos os resultados possíveis de ocorrência do experimento (espaço amostral).

São exemplos de experimentos aleatórios:

- Lançar um dado e observar qual número sairá em sua face voltada para cima.
- Selecionar bolas em uma urna e verificar as suas respectivas cores.

Espaço amostral

É o conjunto de todos os resultados possíveis de ocorrência do experimento aleatório, representado por Ω ou S. São exemplos de espaço amostral:

- No experimento aleatório de lançar um dado e observar qual número sairá em sua face voltada para cima, o espaço amostral associado é S = {1,2,3,4,5,6}.
- No experimento aleatório de selecionar bolas em uma urna e verificar as suas respectivas cores, o espaço amostral associado

é S = {“Branca”,” Preta”}.

Evento

É um subconjunto do espaço amostral. Podem existir diversos tipos de eventos:

- No experimento aleatório de lançar um dado e observar qual número sairá em sua face voltada para cima, o espaço amostral associado é S = {1,2,3,4,5,6} e o evento associado(A) - Faces Pares do dado é {2, 4, 6}.
- No experimento aleatório de selecionar bolas em uma urna e verificar as suas respectivas cores, o espaço amostral associado é S = {“Branca”,” Preta”} e o evento Associado(B) - bola de cor branca é {“Branca”}.

Veremos a seguir como gerar um dado no R por meio da função `rolldie()` da biblioteca **prob** e selecionar o evento de faces pares, por meio do operador `%%` , que nos traz o resto da divisão. Primeiramente, vamos instalar a biblioteca **prob**:

```
# Imports de bibliotecas
install.packages("prob")
```

Em seguida, vamos gerar o dado:

```
# Carregando a biblioteca
library(prob)
# Criando um dado de 6 faces a ser rodado 1 vez, makespace=T cria uma coluna com as probabilidades de cada face
dado = rolldie(1, nsides = 6, makespace = T)
# Selecionando só as faces pares (o operador %% representa o resto da divisão)
subset(dado, X1%%2==0)
```

A saída resultante será:

```
>   X1     probs
> 2  2 0.1666667
```

```
> 4  4 0.1666667
> 6  6 0.1666667
```

Um código similar em Python seria o seguinte:

```
# Criando um dado de 6 faces a ser
dado = [x for x in range(1,7)]
# Selecionando só as faces pares (o operador % representa o resto da divisão)
[x for x in dado if x%2==0]
```

A saída resultante será:

```
[2, 4, 6]
```

A existência do evento está associada à ocorrência de um valor pertencente ao espaço amostral do experimento aleatório. Por exemplo, se a face par de um dado ocorreu, dizemos que o evento A ocorreu.

Existem alguns eventos especiais, a saber:

- **Eventos Certos:** a ocorrência do espaço amostral é chamada de evento certo.
- **Eventos Impossíveis:** a ocorrência do conjunto vazio (*emptyset*) é chamada de evento impossível.
- **Evento Complementar:** é o evento oposto/contrário a determinado evento.
- **Evento União(∪):** quando ocorre um evento OU outro.
- **Evento Intersecção (∩):** quando mais de um evento ocorre simultaneamente.
- **Eventos Mutuamente Exclusivos (Disjuntos):** a ocorrência de um impede a ocorrência do outro, portanto não possuem ponto em comum, ou seja: A ∩ B=\emptyset.
- **Eventos Independentes:** a ocorrência de um não modifica a probabilidade de ocorrência do outro, ou seja: P(A|B) = P(A).

Vejamos um exemplo dos eventos União e Intersecção. Sejam os eventos A e B, definidos como:

- A: "Aparecer faces pares do dado": {2,4,6}.
- B: "Aparecer faces maiores que 3": {4,5,6}.

A **união** dos eventos A e B é dada por: {2,4,5,6}, ou seja, se o evento A ou (inclusivo) o evento B ocorrem, significa dizer que abarcamos todos os elementos de A e B. Já a **intersecção** dos eventos A e B é dada por: {4,6}, ou seja, se o evento A e o evento B ocorrem simultaneamente, significa dizer que abarcamos somente os valores comuns aos dois eventos.

Vejamos no R como calcular a união e intersecção entre os eventos A e B, por meio dos comandos `union` e `intersect` da biblioteca **prob**:

```
# Carregando a biblioteca
library(prob)
# Criando os eventos
A = subset(rolldie(1,makespace = F), X1%%2==0)
B = subset(rolldie(1,makespace = F), X1>3)
# União dos eventos A e B
uniao = union(A,B)
print(uniao)
# Interseção dos eventos A e B
intersecao = intersect(A,B)
print(intersecao)
```

A saída resultante será:

```
> [1] 2 4 5 6
> [1] 4 6
```

O código equivalente em Python seria o seguinte:

```
# Criando os eventos
A = [x for x in dado if x%2==0]
B = [x for x in dado if x>3]
# União dos eventos A e B
uniao = set(A).union(set(B))
print(uniao)
# Interseção dos eventos A e B
```

```
intersecao = set(A).intersection(set(B))
print(intersecao)
`
```

A saída resultante será:

```
{2, 4, 5, 6}
{4, 6}
```

4.2 Axiomas de probabilidade de Kolmogorov

Axiomas são verdades que não precisam ser provadas, princípios base de uma teoria. Os axiomas que serão apresentados a seguir são atribuídos ao russo Kolmogorov. Toda probabilidade é uma função que satisfaz os três axiomas de Kolmogorov:

- **Primeiro axioma de Kolmogorov:** toda probabilidade é limitada inferiormente em 0 e superiormente em 1, para o seu espaço amostral associado, ou seja, 0 `<=` P(A) `<=` 1, para todo A definido em S.
- **Segundo axioma de Kolmogorov:** a ocorrência do espaço amostral é chamada de **Evento Certo**, ou seja, P(S)=1. A ocorrência do conjunto vazio `(\emptyset)` é chamada de **Evento Impossível**, ou seja, `P(\emptyset)=0` .
- **Terceiro axioma de Kolmogorov:** se existem eventos que não possuem intersecção, a probabilidade da união entre esses eventos será dada pela soma das probabilidades dos mesmos, ou seja: sejam A1, A2, ... eventos em S, tais que:

$$\cap_{i=1}^{\infty} A_i = \emptyset$$

Então:

$$P(\cup_{I=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$$

4.3 Atribuição das probabilidades

Para elementos do espaço amostral com equiprobabilidade (mesma probabilidade de acontecerem), podemos usar esta abordagem:

$$\mathbf{P(A)} = \frac{\#A}{\#B} = \frac{"Casos\ Favoráveis"}{"Casos\ Possíveis"} = \frac{"Quero"}{"Total"}$$

Exemplo

Seja o evento "obter duas caras na face de uma moeda voltada para cima, lançando duas moedas", calcule a probabilidade da ocorrência do evento, "ocorrerem duas caras".

Vamos definir cada elemento do problema:

- Experimento aleatório: "lançar as duas moedas e observar o resultado da face voltada para cima."
- Evento associado: "obter duas caras no lançamento de duas moedas".
- Espaço amostral ("Casos Possíveis"/"Total"): {(ca, ca, ca), (ca, ca, co), (ca, co, ca), (co, ca, ca), (ca, co, co), (co, ca, co), (co, co, ca), (co, co, co)} (8 possíveis)
- Casos favoráveis: {(ca, ca, co), (ca, co, ca), (co, ca, ca)} (3 favoráveis)

P(A) = (#A)/(#B) = 3/8 = 0,375 ou 37,5%

Vejamos no R como calcular essa probabilidade utilizando a função `tosscoin()` da biblioteca **prob** e os comandos `dim()`, que trazem o **número de linhas e colunas** de um **Dataframe**:

```r
# Carregando a biblioteca
library(prob)
# Sendo H do inglês "Head" que significa "cara"
# Sendo T do inglês "Tail" que significa "coroa"
# Cria e imprime o espaço amostral para um lançamento de moedas
moedas = tosscoin(3, makespace = F)
print(moedas)
# Selecionando as linhas 2, 3 e 5 (que têm duas caras) e todas as colunas
do dataframe moedas
duas_caras = moedas[c(2,3,5),]
# Calculando a probabilidade desejada
dim(duas_caras)[1]/dim(moedas)[1]
```

A saída resultante será:

```
>   toss1 toss2 toss3
> 1     H     H     H
> 2     T     H     H
> 3     H     T     H
> 4     T     T     H
> 5     H     H     T
> 6     T     H     T
> 7     H     T     T
> 8     T     T     T
> [1] 0.375
```

O código equivalente em Python seria o seguinte:

```python
# Imports necessários
import math
import itertools
import pandas as pd
from collections import Counter

# Sendo H do inglês "Head" que significa "cara"
# Sendo T do inglês "Tail" que significa "coroa"
# Cria e imprime o espaço amostral para um lançamento de moedas
qtd_moedas = 3
moedas = [x for x in ("H","T")] # H=head(cara) e T=tail(coroa)
# Número de combinações possíveis
```

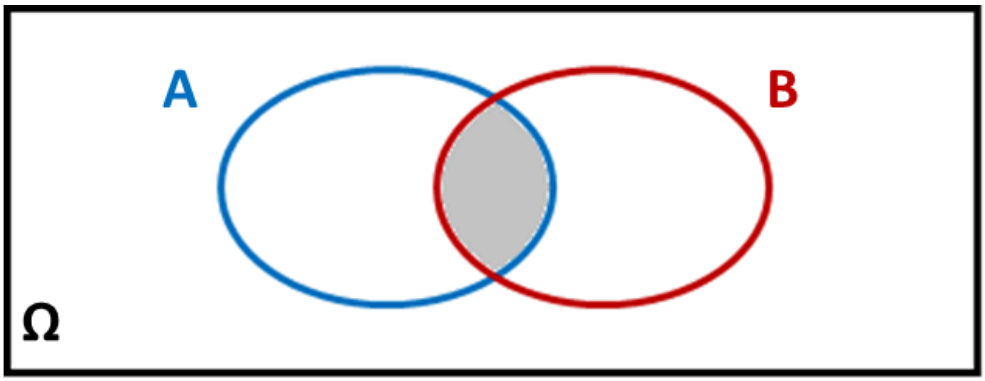
```
comb = int(math.pow(len(moedas),qtd_moedas))

# Cria matriz vazia
mres = [[0] * qtd_moedas for i in range(comb)]

# Combinações possíveis
iterables = [ ["H","T"], ["H","T"], ["H","T"] ]
for t, i in zip(itertools.product(*iterables), range(0,comb)):
    mres[i] = t

mres = pd.DataFrame(mres, columns = ["M1","M2","M3"])
print(mres)

# Selecionando as linhas que têm duas caras
mres["Qtd_H"] = mres.apply(lambda row: Counter(row)["H"], axis = 1)
# Calculando a probabilidade desejada
p2h = mres[mres["Qtd_H"]==2].shape[0]/mres.shape[0]
print(p2h)
```

E o resultado:

```
  M1 M2 M3
0  H  H  H
1  H  H  T
2  H  T  H
3  H  T  T
4  T  H  H
5  T  H  T
6  T  T  H
7  T  T  T
0.375
```

Probabilidade da união entre dois eventos

Sejam os eventos A e B, dispostos no diagrama de Venn:



$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

Observação: note que, ao fazer a união entre os eventos A e B, eles possuem pontos em comum, sendo contados duas vezes. Assim, há a necessidade de se retirar a intersecção entre eles, para não haver duplicidade.

Exemplo

Um aluno estuda para um exame por dois livros. O primeiro aborda 30% do programa, o segundo 28%, e 24% é abordado por ambos os livros. Qual a probabilidade de que determinado tópico seja abordado em **pelo menos um dos dois livros** utilizados pelo aluno?

Vamos primeiro definir os eventos:

- Evento A = "Tópico estar no primeiro livro"
- Evento B = "Tópico estar no segundo livro"

Sabemos pelo enunciado que:

$P(A)=0,30$
$P(B)= 0,28$
$P(A \cap B)= 0,24$

Então a união entre os eventos será dada por:

$P(A \cup B) = P(A)+P(B)-P(A \cap B)$.

Substituindo: $P(A \cup B) = 0,30 + 0,28 – 0,24 = 0,34$ ou 34%.

Vejamos este exemplo em código. O código a seguir é idêntico tanto para R quanto para Python:

```
# Criando o objeto P(A)
PA = 0.30
# Criando o objeto P(B)
PB = 0.28
# Criando o objeto contendo a probabilidade da interseção
PAinterB = 0.24
# Calculando a Probabilidade da União
PAuniaoB = PA + PB - PAinterB
print(PAuniaoB)
```

Saída:

```
> [1] 0.34
```

Probabilidade da intersecção entre dois eventos

Perceba que basta somente isolar o $P (A \cap B)$ na fórmula da probabilidade da união para se chegar à fórmula da probabilidade da intersecção:

$P (A \cap B) = P(A)+P(B)- P (A \cup B)$.

Exemplo

Um aluno estuda para um exame por dois livros. O primeiro aborda 30% do programa, o segundo 28%, e 34% é abordado pelo primeiro ou pelo segundo. Qual a probabilidade de que determinado tópico seja abordado em **ambos** os livros utilizados pelo aluno?

Vamos primeiro definir os eventos:

- Evento A = “Tópico estar no primeiro livro”
- Evento B = “Tópico estar no segundo livro”

Sabemos pelo enunciado que:

```
P(A)=0,30
P(B)= 0,28
P(A ∪ B)= 0,44
```

Então, a união entre os eventos será dada por: $P(A \cup B) = P(A)+P(B)-P(A \cap B)$.

Substituindo: $P(A \cap B) = 0,30 + 0,28 – 0,34 = 0,24$ ou 24%.

Vejamos este exemplo em código. O código a seguir é idêntico tanto para R quanto para Python:

```
# Criando o objeto P(A)
PA = 0.30
# Criando o objeto P(B)
PB = 0.28
# Criando o objeto contendo a probabilidade da união
PAuniaoB = 0.24
# Calculando a Probabilidade da União
PAInterB = PA + PB - PAuniaoB
print(PAuniaoB)
```

Saída:

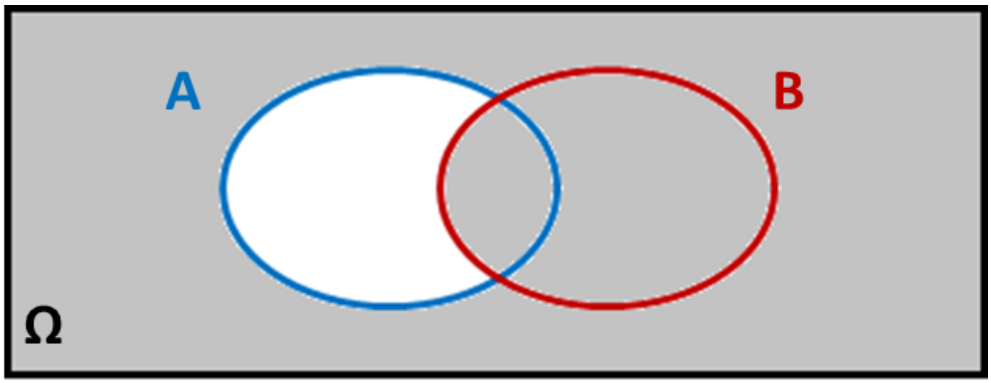
```
> [1] 0.24
```

Teoremas de DeMorgan

Os teoremas de De Morgan são propostas de simplificação de expressões matemáticas que definem regras usadas para converter operações lógicas. Sejam:

- A^C = Evento Complementar de A.
- B^C = Evento Complementar de B.

A título de ilustração, o evento complementar de A é ilustrado pela figura a seguir:



Primeira Lei de Morgan: $A^C \cap B^C = (A \cup B)^C$

Segunda Lei de Morgan: $A^C \cup B^C = (A \cap B)^C$

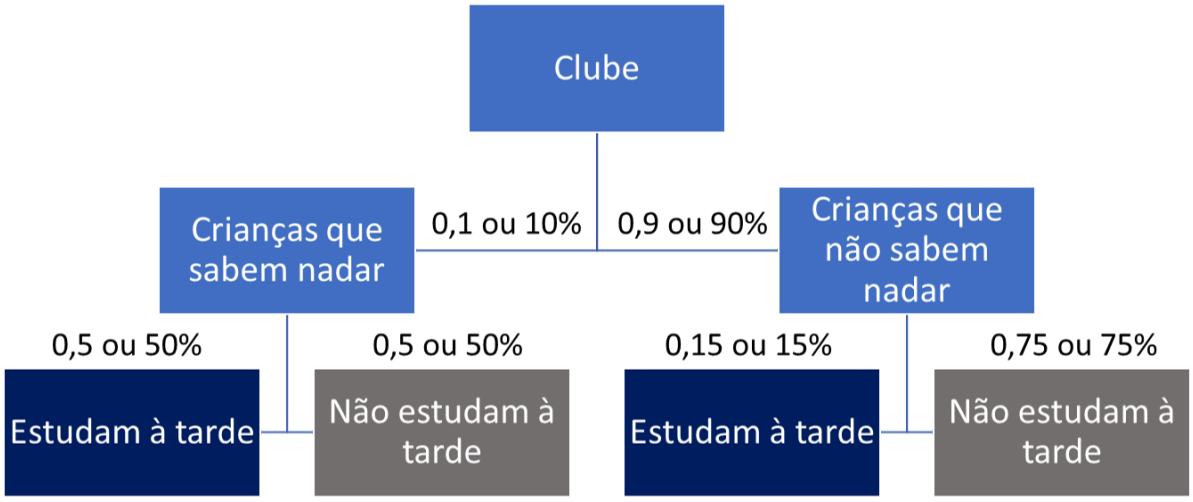
Lei da probabilidade total

Consiste em contemplar todos os caminhos possíveis para se chegar ao evento A:

$P(A) = P(A/B)\ P(B) + P(A/B^C) \times P(B^C)$

Exemplo

Em um clube de natação, existe um grupo de crianças em que apenas 10% sabem nadar. Dentre as crianças que sabem nadar, 50% estudam à tarde, enquanto, dentre aquelas que não sabem nadar, 15% estudam à tarde. Em relação ao grupo todo, qual é o percentual de crianças que estudam à tarde?



Percebemos então que existem dois caminhos possíveis para determinar o número de crianças que estudam à tarde:

Portanto, basta somarmos: 0,05 + 0,135 = 0,185 ou 18,5%.

Sejam os eventos A = "A criança saber nadar" e B = "A criança estudar à tarde", vejamos no R como calcular essa probabilidade (o código em Python é idêntico):

```
# Criando o objeto P(A)
PA = 0.10
# Criando o objeto P(AC), Evento complementar ao A
PAC = 0.9
# Criando o objeto PBdadoA
PBdadoA = 0.5
```

```
# Calculando o objeto PBCdadoA
PBCdadoA = 0.5
# Calculando o objeto PBdadoAC
PBdadoAC = 0.15
# Calculando o objeto PBCdadoAC
PBCdadoAC = 0.85
#Queremos saber PB
PB = PA * PBdadoA + PAC * PBdadoAC
print(PB)
```

Saída:

```
> [1] 0.185
```

Probabilidade condicional

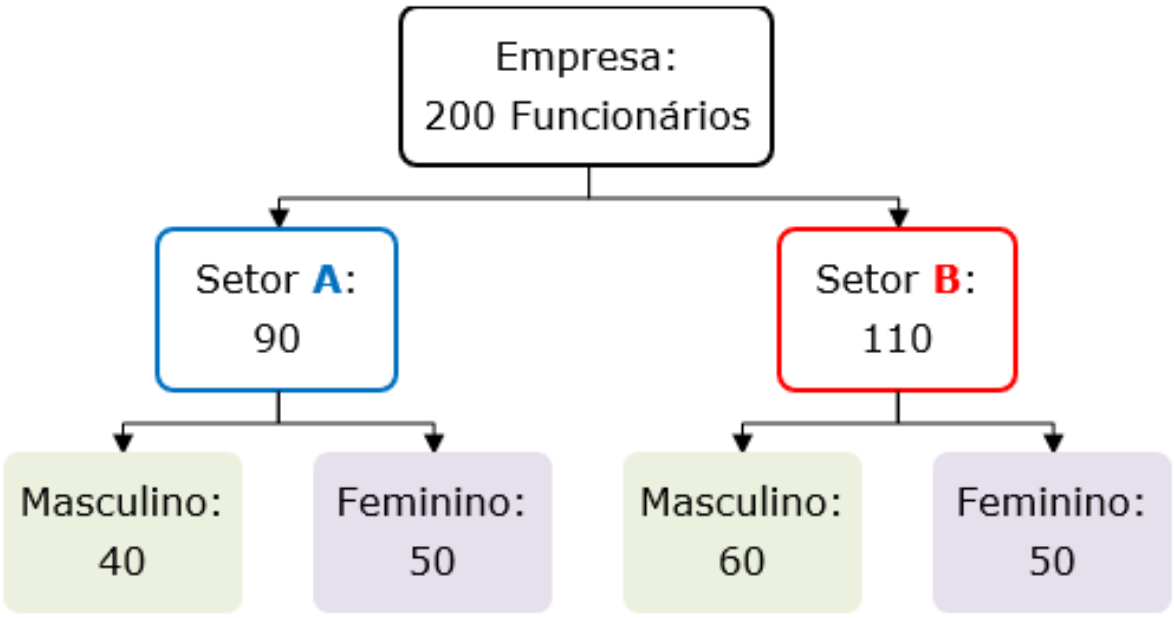
Quando um determinado evento é condicionado, seu espaço amostral é reduzido por uma informação adicional que o limita, ou seja, há uma restrição do seu espaço amostral. Essa condição é expressa pelo teorema de Bayes:

$$\mathbf{P(A|B)} = \frac{\boldsymbol{P(A \cap B)}}{\mathbf{P(B)}}$$

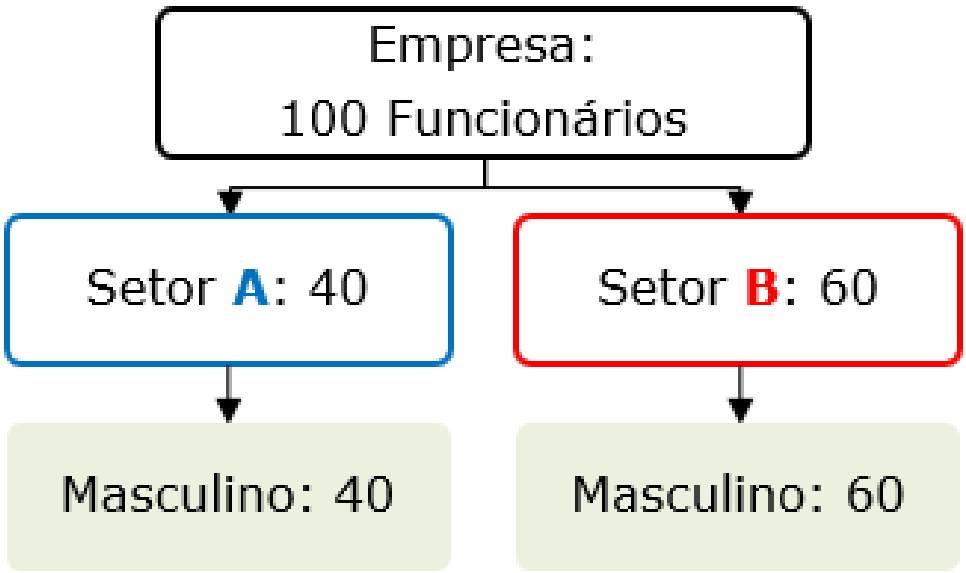
Exemplo 1

Em uma empresa existem 100 funcionários do sexo masculino, dos quais 40 trabalham no setor A e 60 no setor B. Nesta mesma empresa, existem 100 funcionários do sexo feminino, das quais 50 trabalham no setor A e 50 no setor B. Dos funcionários do sexo masculino, qual a probabilidade de escolhermos um do setor A para ganhar um presente?

Vamos esquematizar para gerar um melhor entendimento:



Temos um caso de probabilidade condicional, pois só estamos interessados em funcionários do sexo masculino, ou seja, o espaço amostral fica condicionado a somente esses funcionários. Vamos ilustrar como fica nosso esquema agora:

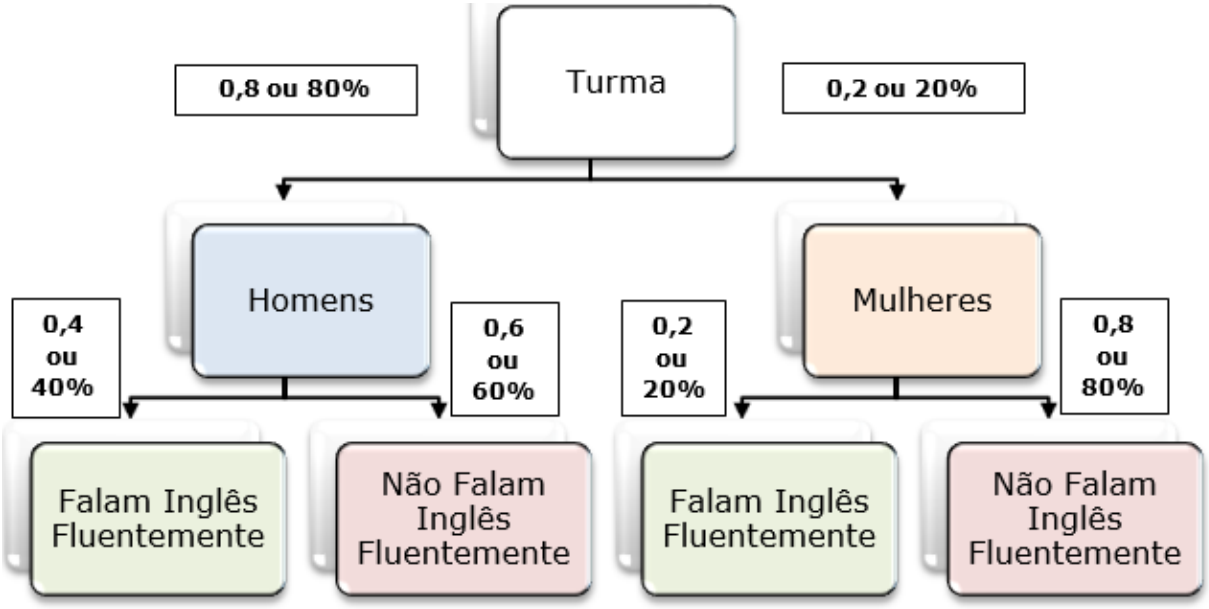


Evento A: "Ser funcionário do setor A". Evento M: "Ser funcionário do Sexo Masculino".

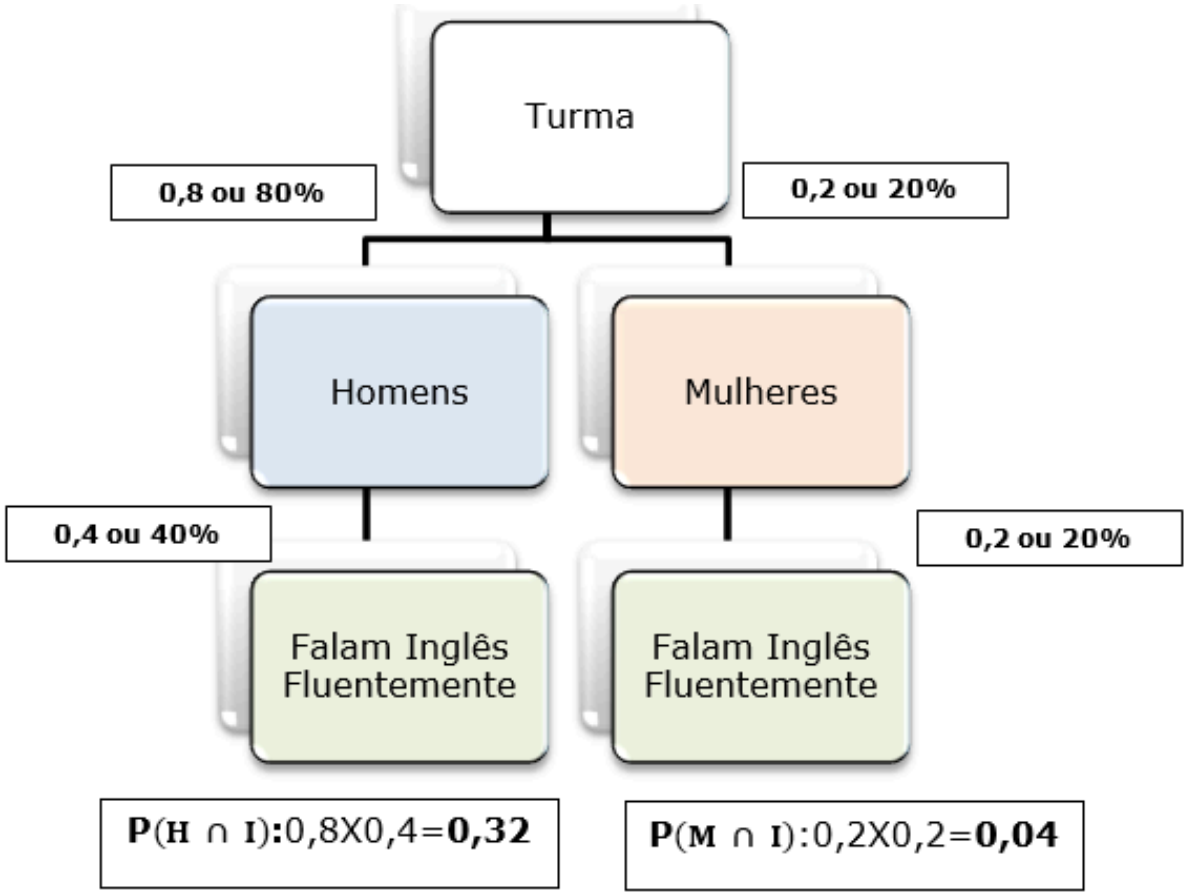
$$\mathbf{P(A|M)} = \frac{P(A \cap M)}{\mathbf{P(B)}} = \frac{40}{100} = \mathbf{0{,}4 \text{ ou } 40\%.}$$

Exemplo 2

Em certa turma, 40% dos homens e 20% das mulheres falam inglês fluentemente. 80% das pessoas são homens. Qual é a probabilidade de um aluno fluente na língua inglesa, selecionado ao acaso, ser homem? Vamos esquematizar para gerar um melhor entendimento:



Quando foi perguntado "sabendo que o aluno é fluente na língua inglesa, qual a probabilidade de selecionar um aluno do sexo masculino" foi introduzida uma informação adicional, que limita o nosso espaço amostral aos alunos que falam inglês fluentemente. Nosso esquema fica então da seguinte forma:



$$P(H|I) = \frac{P(H \cap I)}{\textit{Lei da Probabilidade Total}}$$

Lei da probabilidade Total= $P(H \cap I) + P(M \cap I) = \mathbf{0,32+0,04=0,36}$

Substituindo, ficamos com:

$$P(H|I) = \frac{0,32}{0,36} = \frac{32/100}{36/100} = \frac{16}{18} = \frac{8}{9} \approx \mathbf{0,89\ ou\ 89\%}$$

Seja o evento A = "ser homem" e o evento B = "falar inglês fluentemente", no R temos (o código em Python é idêntico):

```
# Calculando PAinterB
PAinterB = 0.8 * 0.4
# Calculando PAinterB
PACinterB = 0.2 * 0.2
# Calculando PB
```

```
PB= PAinterB + PACinterB
# Utilizando a fórmula de Bayes
bayes = PAinterB/PB
print(bayes)
```

Sendo a saída:

```
> [1] 0.8888889
```

Este capítulo apresentou a primeira parte do conteúdo referente a cálculo de probabilidades, abordando conceitos fundamentais em probabilidade: axiomas de Kolmogorov, probabilidade da união e intersecção, leis de Morgan, lei da probabilidade total, probabilidade condicional e teorema de Bayes. O próximo capítulo apresentará a segunda parte do conteúdo referente a cálculo de probabilidades, abordando distribuições discretas e contínuas.

Capítulo 5
Cálculo das Probabilidades: Distribuições de probabilidade discretas e contínuas

No capítulo anterior, estudamos conceitos e fundamentos sobre o cálculo de probabilidades e vimos como calcular a probabilidade de um evento. Mas também podemos calcular a probabilidade de todos os resultados possíveis. Uma **variável aleatória** é uma representação numérica dos resultados possíveis de um experimento aleatório. Os tipos de variáveis aleatórias são ilustrados na figura a seguir:

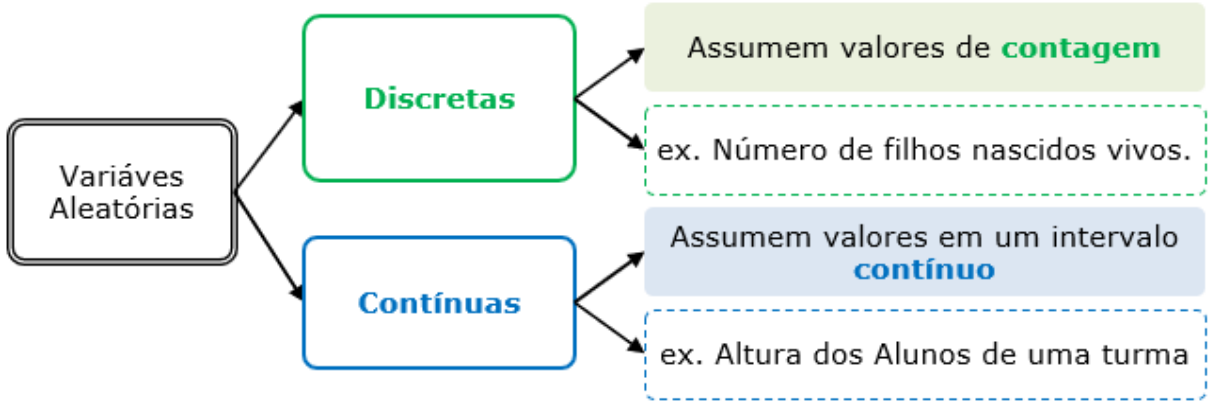


Figura 5.1: Tipos de variáveis aleatórias

Assim, uma variável aleatória discreta tem um conjunto finito de valores, enquanto uma variável aleatória contínua tem um intervalo de valores numéricos. Uma variável aleatória geralmente é representada por uma letra maiúscula (por exemplo, X) e os seus possíveis valores, por uma letra minúscula e um índice numérico (por exemplo, x_1, x_2 e x_3). A probabilidade de uma variável aleatória é representada por P(X), quando quisermos nos referir à probabilidade de todos os valores para a variável aleatória X, e por P(X = 5), quando quisermos indicar a probabilidade de a variável aleatória X ter o valor 5.

Uma **distribuição de probabilidade** resume as probabilidades para os valores possíveis de uma variável aleatória, ou seja, a relação entre cada resultado possível para uma variável aleatória e suas probabilidades. A estrutura e o tipo da distribuição de probabilidade variam com base nas propriedades da variável aleatória (por exemplo, se ela é contínua ou discreta). Isso impacta em como a distribuição pode ser resumida e em como calcular o seu resultado mais provável e sua probabilidade. A notação para dizer que uma variável aleatória X segue uma determinada distribuição de probabilidade é: `X ~ <distribuição>`

O **valor esperado** e a **variância** são duas propriedades importantes de uma distribuição de probabilidade, conhecidas matematicamente como o primeiro momento e o segundo momento centrado da distribuição, respectivamente. O valor esperado `(E[X])` é o valor médio de uma variável aleatória X, que é o seu valor mais provável, isto é, o resultado com maior probabilidade. Já a variância `(Var(X))` é a dispersão dos valores de uma variável aleatória a partir da média. Já vimos que a raiz quadrada da variância é o **desvio padrão**. A variância entre duas variáveis é chamada de **covariância** e indica como duas variáveis aleatórias mudam juntas.

5.1 Distribuições de probabilidade discretas

As distribuições de probabilidade discretas sumarizam as probabilidades de uma variável aleatória discreta, ou seja, descrevem a probabilidade de ocorrência de cada valor de uma variável aleatória discreta (valores finitos e enumeráveis) através de uma função matemática que atribui uma probabilidade de saída a cada valor discreto específico, conhecida como **função massa de probabilidade**.

No caso de um dado não viciado, os valores possíveis em um lançamento são 1, 2, 3, 4, 5 ou 6, que representam os valores

possíveis da variável aleatória discreta, e a função massa de probabilidade associará a cada uma dessas ocorrências uma probabilidade igual a 1/6, sendo 1 a soma de todas as probabilidades.

Já a **função de distribuição acumulada** descreve completamente a distribuição da probabilidade de uma variável aleatória discreta, retornando a probabilidade de essa variável ter um valor menor ou igual a um valor discreto de saída específico. Ou seja, esta função é igual à soma dos valores de todas as probabilidades até um certo ponto.

Em Ciência de Dados, as distribuições de probabilidade discretas são muito utilizadas em Machine Learning, por exemplo, para:

- Modelar e avaliar o desempenho de problemas de classificação;
- Modelar a distribuição de palavras em um texto, em problemas de processamento de linguagem natural
- Escolher as funções de ativação na camada de saída de redes neurais.

As principais distribuições de probabilidade discretas são **Bernoulli**, **Binomial** e **Poisson**, que serão detalhadas a seguir.

Distribuição de Bernoulli

É um experimento aleatório com dois resultados possíveis (fracasso ou sucesso) e efetuado em uma única realização. Exemplos: jogar uma moeda uma vez para cima e ver se sai cara ou coroa; determinar se uma empresa terá que pagar um determinado imposto ou não. Observe a tabela a seguir:

X	P(X=x)
0	1-p
1	p

A variável aleatória *X* está assumindo 0 (fracasso), com probabilidade *q* ou *1-p* e assumindo 1 (sucesso), com probabilidade *p*.

Imaginemos que X = "O aluno acertar uma questão na prova", sendo que a prova só tem uma questão, podendo ser respondida de forma correta ou errada. Podemos entender que o aluno tem uma resposta favorável em duas possíveis, ou seja, 50% de chance de acertar a questão.

Vamos chamar de *p* a probabilidade de sucesso, isto é, ele acertar a questão, e de *q* ou *1-p* a probabilidade de fracasso, isto é, ele errar a questão. Nesse caso então $p = 0,5$ e $q = 0,5$ também. Logo, concluímos que $P(X=x) = 0,5^x(0,5)^{1-x}$, onde $x = 0,1$ e $0 < p < 1$.

Queremos que ele acerte uma questão em uma. Então *x* é o número de sucessos que queremos, logo $x = 1$.

$P(X=1) = 0,5$ ou 50%

Vejamos no R como calcular essa probabilidade, utilizando as funções da biblioteca base `dbinom()` e `pbinom()`, utilizando os argumentos **"número de sucessos (x)", "quantidade de tentativas (size)" e "probabilidade de sucesso (p)"**:

```
# Calculando pela função de densidade de probabilidade aplicando para X=1: P(X=1)
print(dbinom(1,size=1,p=0.5)) # Resultado 0.5
# Calculando pela funçao de distribuição acumulada, fazendo o equivalente a P(X<=1)-P(X<=0)= P(X=1), que é o que queremos calcular.
print(pbinom(1,size=1,p=0.5)-pbinom(0,1,p=0.5))  # Resultado 0.5
```

E o equivalente em Python, usando a função `binom()`:

```
# Imports de bibliotecas
from scipy.stats import binom
# Calculando pela função de densidade de probabilidade aplicando para X=1: P(X=1)
print(binom.pmf(1, 1, 0.5)) # Resultado 0.5
```

```
# Calculando pela funçao de distribuição acumulada, fazendo o equivalente
a P(X<=1)-P(X<=0)= P(X=1), que é o que queremos calcular.
print(binom.cdf(1, 1, 0.5)-binom.cdf(0, 1, 0.5)) # Resultado 0.5
```

Dizemos que $X \sim Bernoulli(p)$ e sua fórmula é dada genericamente por:

$P(X=x)=p^x(1-p)^{1-x}$, *onde* $x = 0,1$ *e* $0 < p < 1$

onde:

- Valor Esperado / Esperança / Média = p
- Variância = pq ou $p(1-p)$

Distribuição Binomial

Representa n realizações independentes de experimentos Bernoulli, com a mesma probabilidade de sucesso p. Exemplos: jogar uma moeda, mais de uma vez, e ver quantas caras e coroas sairão; determinar se uma empresa deverá pagar mais de uma vez determinado imposto. A distribuição Bernoulli pode ser considerada uma distribuição binomial com uma única tentativa.

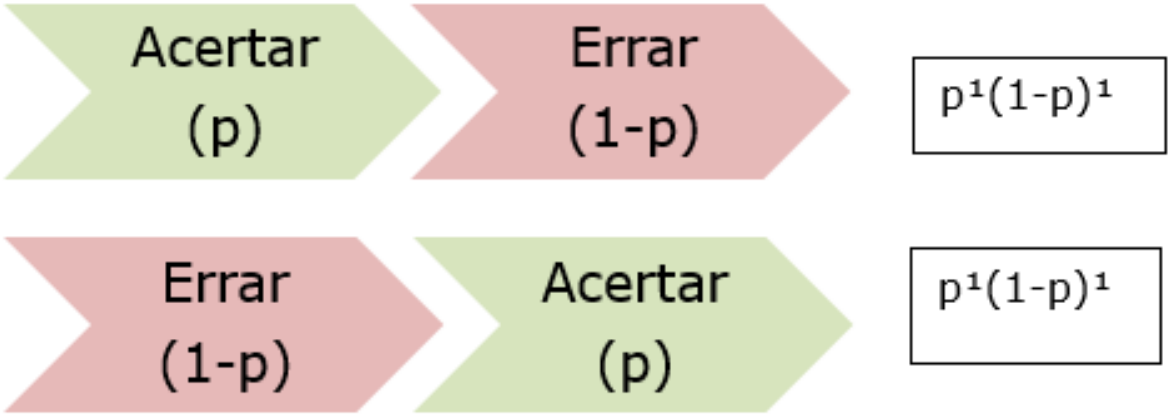
Vejamos a tabela a seguir:

X	P(X=x)	X	P(X=x)
0	1-p	1	P
1	P	0	1-p

A variável aleatória X está assumindo 1 (sucesso), com probabilidade p e e assumindo 0 (fracasso), com probabilidade $1-p$ (ou q ou), mas agora a mesma variável X tem duas realizações distintas.

No exemplo anterior, de o aluno acertar uma questão na prova que tem duas questões, vimos que o aluno tem uma resposta favorável em duas possíveis (50% de chance de acertar cada questão). Vamos chamar então de p, a probabilidade de sucesso, ou seja, ele acertar

a questão, e de *q* ou *1-p* a probabilidade de fracasso, ou seja, ele errar a questão. Nesse caso então *p = 0,5* e *q = 0,5* também. Entretanto, diferentemente do exemplo anterior, o aluno tem as possibilidades de acertar só uma questão, ambas as questões ou nenhuma questão, e acertar a primeira questão é independente de acertar a segunda e vice-versa:



Logo, concluímos que ele pode acertar tanto na primeira e errar a segunda, como acertar a segunda e errar a primeira, acertar ambas as questões ou nenhuma. Todas as possibilidades devem ser levadas em consideração, portanto, como a ordem importa, o resultado será a combinação de *n* elementos tomados *x* a *x*. Neste caso, a combinação de 2 elementos tomados 1 a 1.

Ficamos com:

$$P(X = x) = \binom{n}{x} 0{,}5^{x}(0{,}5)^{2-x}, onde\ x = 0, 1, 2\ e\ 0 < p < 1$$

onde:

- *X* é o número de sucessos que queremos, logo 1.
- *n* é a quantidade de tentativas possíveis, logo 2, pois são duas questões.

Assim:

$$P(X = 1) = \binom{2}{1} 0{,}5^1 (0{,}5)^{2-1} = 0{,}5 \; ou \; 50\%$$

Vejamos no R como calcular essa probabilidade, utilizando novamente as funções `dbinom()` e `pbinom()`:

```
# Calculando pela função de densidade de probabilidade aplicando para X=1: P(X=1)
print(dbinom(1,size=2,0.5)) # Resultado 0.5
# Calculando pela funçao de distribuição acumulada, fazendo o equivalente a P(X<=1)-P(X<=0)= P(X=1), que é o que queremos calcular.
print(pbinom(1,size=2,p=0.5)-pbinom(0,2,p=0.5)) # Resultado 0.5
```

E o equivalente em Python:

```
# Imports de bibliotecas
from scipy.stats import binom
# Calculando pela função de densidade de probabilidade aplicando para X=1: P(X=1)
print(binom.pmf(1, 2, 0.5)) # Resultado 0.5
# Calculando pela funçao de distribuição acumulada, fazendo o equivalente a P(X<=1)-P(X<=0)= P(X=1), que é o que queremos calcular.
print(binom.cdf(1, 2, 0.5)-binom.cdf(0, 2, 0.5)) # Resultado 0.5
```

Dizemos que *X ~ Binomial(n, p)* e sua fórmula é dada genericamente por:

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}, onde \; x = 0, 1, 2, \ldots, n \; e \; 0 < p < 1$$

onde:

- Valor Esperado / Esperança / Média = *np*
- Variância = *npq* ou *np(1-p)*

Observações: note também que a Bernoulli é um caso particular de Binomial, com *n=1*. Assumindo *n=1* na fórmula da Binomial, chegamos tanto à fórmula da Bernoulli, quanto à sua esperança e variância.

Exemplo

Considere um exame de múltipla escolha com 20 questões e 5 alternativas para cada pergunta. Caso o aluno não estude e "chute" todas as respostas, qual a probabilidade de acertar 30% da prova? E qual seu número esperado de acertos?

Temos:

- n = 20, número de questões (tentativas)
- X = "O aluno acertar uma questão"
- p = 1/5 ou 20%
- q = 1-p = 4/5 ou 80%
- X~Binomial(20,1/5)

Queremos que ele acerte 30% de 20, o que significa acertar 6 questões. Logo, precisamos calcular a probabilidade de *X = 6*. Substituindo na fórmula:

$$P(X=6) = \binom{20}{6} 0,2^6 0,8^{14} = 0,109 \text{ ou } 10,9\%$$

O valor esperado da Binomial é E(x) = n.p = 20.0,2 = 4 acertos em média.

Vejamos no R como calcular essa probabilidade, utilizando as funções da biblioteca base `dbinom()` e `pbinom()`:

```
n = 20 # número de questões da prova
p = 0.2 # probabilidade de sucesso (acertar uma questão em cinco)

# Utilizando a função densidade de probabilidade aplicando para X=6:
P(X=6)
```

```
print(dbinom(6,n,p))

# Utilizando a função de distribuição acumulada P(X<=6)-P(X<=5)= P(X=6)
print(pbinom(6,n,p)-pbinom(5,n,p))

# Calculando o valor esperado
valor_esperado = n*p
print(valor_esperado)
```

A saída será:

```
[1] 0.1090997
[1] 0.1090997
[1] 4
```

E o equivalente em Python:

```
# Imports de bibliotecas
from scipy.stats import binom

n = 20
p = 0.2

# Utilizando a função densidade de probabilidade aplicando para X=6:
P(X=6)
print(binom.pmf(6,n,p))

 # Utilizando a função de distribuição acumulada P(X<=6)-P(X<=5)= P(X=6)
print(binom.cdf(6,n,p)-binom.cdf(5,n,p))

# Calculando o valor esperado
valor_esperado = n*p
print(valor_esperado)
```

A saída será:

```
0.10909970097304993
0.10909970097305033
4.0
```

Em Machine Learning, a predição em um problema de classificação binária pode ser vista como uma tentativa de Bernoulli (correta ou incorreta). A distribuição binomial resume a acurácia do classificador (probabilidade de acertos) para as *n* instâncias (equivalentes ao número de tentativas de Bernoulli) do problema.

Distribuição de Poisson

Representa números de ocorrências de um evento em um intervalo correspondente. Exemplos: número de carros passando em uma ponte, número de pessoas chegando em uma fila, número de navios atracando no porto.

Dizemos que X ~ Poisson(λ) e sua fórmula é dada por:

$$P(X = x) = \frac{\lambda^x e^{-\lambda}}{x!}, x = 0, 1, \ldots; \lambda > 0$$

onde:

- Valor Esperado = $E(X) = \lambda$
- Variância = $V(X) = \lambda$

> Observação: a distribuição de Poisson é a única distribuição em que a Média ou Valor Esperado é igual à Variância.

Vejamos como utilizar a distribuição de Poisson no exemplo a seguir.

Exemplo

O número de acidentes que acontecem na ponte Rio-Niterói segue uma distribuição de Poisson com média 3 por hora. Pede-se:

a) Calcule a probabilidade de 2 acidentes em uma hora.

$$P(X = 2) = \frac{3^2 e^{-3}}{2!} = 4,5e^{-3} = 0,22404 \text{ ou } 22,404\%$$

b) Calcule a probabilidade de pelo menos 2 acidentes em 2 horas.

Como queremos para duas horas e nosso λ é 3 por hora, vamos fazer a regra de 3 para achar nosso novo λ:

- 3 – 1 hora
- x – 2 horas

x = 6, portanto nosso novo λ é 6.

"Pelo menos 2" significa que são 2 ou mais, então:

$$P(x \geq 2) = 1 - \big(P(X = 0) + P(X = 1)\big) = \frac{6^0 e^{-6}}{0!} + \frac{6^1 e^{-6}}{1!} = 1 - \left(e^{-6}(1 + 6)\right) = 1 - 7e^{-6} = 0,9826 \text{ ou } 98,26\%.$$

Vejamos no R como calcular essa probabilidade, utilizando as funções da biblioteca base `dpois()` e `ppois()`:

```
# a) Como o a taxa solicitada está em horas, manteremos o nosso lambda
lambda = 3

# Utilizando a função densidade de probabilidade aplicando para X=2: P(X=2)
print(dpois(2,lambda))

# Utilizando a função de distribuição acumulada P(X<=2) - P(X<=1) = P(X=2)
print(ppois(2,lambda)-ppois(1,lambda))

# Fazendo o equivalente manualmente
print((3^2)*exp(-3)/factorial(2))

# b) Como o a taxa solicitada está em 2 horas, ajustaremos o lambda proporcionalmente para 6
lambda = 6

# Calculando o complementar da probabilidade solicitada ou seja: 1 -
```

```
P(X<=1) = 1-P(X<2) = P(X>=2)
print(1-ppois(1,lambda))
```

A saída será:

```
[1] 0.2240418
[1] 0.2240418
[1] 0.2240418
[1] 0.9826487
```

```
# Imports de bibliotecas
import math
from scipy.stats import poisson

# a) Como a taxa solicitada está em horas, manteremos o nosso lambda
lamb = 3 # OBS: lambda é palavra reservada em Python

# Utilizando a função densidade de probabilidade aplicando para X=2:
P(X=2)
print(poisson.pmf(2,lamb))

# Utilizando a função de distribuição acumulada P(X<=2) - P(X<=1) = P(X=2)
print(poisson.cdf(2,lamb)-poisson.cdf(2-1,lamb))

# Fazendo o equivalente manualmente
print(math.pow(lamb,2)*math.exp(-lamb)/math.factorial(2))

# b) Como a taxa solicitada está em 2 horas, ajustaremos o lambda
proporcionalmente para 6
lamb = 6

# Calculando o complementar da probabilidade solicitada ou seja: 1 -
P(X<=1) = 1-P(X<2) = P(X>=2)
print(1-poisson.cdf(2-1,lamb))
```

A saída será:

```
0.22404180765538775
0.22404180765538784
```

```
0.22404180765538775
0.9826487347633355
```

5.2 Distribuições de probabilidade contínuas

Já sabemos que uma variável aleatória contínua tem um valor numérico real e que cada possível saída pode receber uma probabilidade. Assim, as distribuições de probabilidade contínuas são funções que permitem calcular a probabilidade de que uma variável aleatória contínua pertença a um determinado intervalo, e podem ser resumidas por uma **função de densidade de probabilidade**.

Ao contrário das variáveis aleatórias discretas, a probabilidade de variáveis aleatórias contínuas não pode ser especificada diretamente, e precisa ser calculadas como uma integral (que, graficamente, representa a área sob a curva) para um pequeno intervalo em torno de uma saída específica. De forma análoga às variáveis aleatórias discretas, a **função de distribuição acumulada** retorna a probabilidade de uma variável aleatória contínua ter um valor menor ou igual a um valor contínuo específico.

Em Ciência de Dados, as distribuições de probabilidade contínuas são muito utilizadas em Machine Learning, por exemplo, para:

- Avaliar a distribuição de variáveis numéricas de entrada dos modelos;
- Avaliar a distribuição dos erros gerados pelos modelos;
- Mapear o relacionamento entre entradas e saídas dos modelos.

As principais distribuições de probabilidade contínuas utilizadas em Ciência de Dados são **Uniforme**, **Exponencial** e **Normal**, que serão detalhadas a seguir.

Distribuição Uniforme Contínua ou Retangular

Como o próprio nome diz, é caracterizada por manter a probabilidade constante (uniforme) em todo seu intervalo com mínimo em *a* e máximo em *b*. Ou seja, graficamente, esta probabilidade constante delimita a base de um retângulo e, por este motivo, esta distribuição de probabilidade também é chamada de Retangular.

A distribuição Uniforme de uma variável X pode ser representada pela seguinte fórmula:

$f(x) = 1/(b-a),\ a \leq x \leq b$

$X \sim Uniforme(a,b)$

onde:

- Valor esperado = $E(x) = (b+a)/2$
- Variância = $V(x) = (b-a)^2/12$

A figura a seguir ilustra o formato desta distribuição:

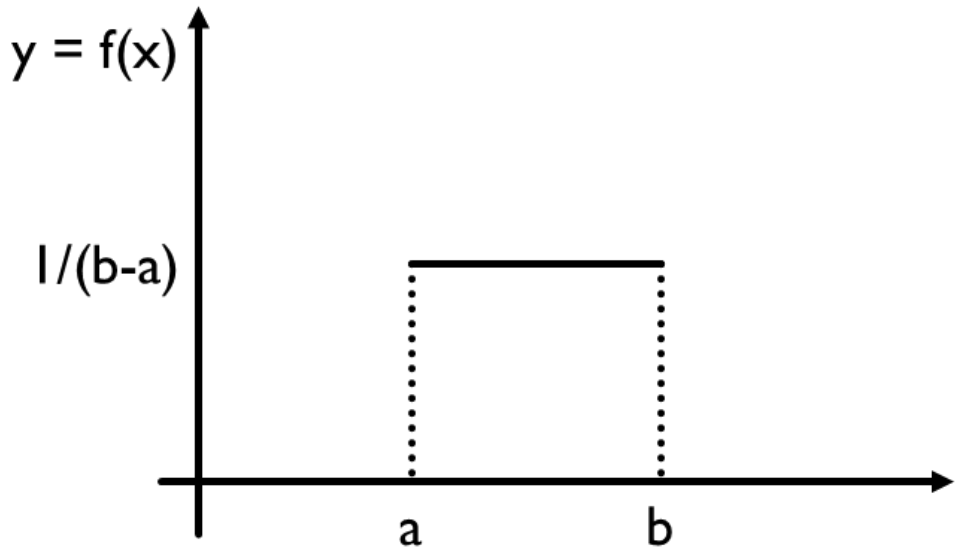


Figura 5.10: Distribuição Uniforme Contínua ou Retangular

Exemplo

Seja a variável aleatória contínua X a corrente medida em um fio delgado de cobre em miliamperes (mA). Considere que a faixa de X possa variar entre 0 e 10 mA. Pede-se:

- a) Calcule a função de densidade de probabilidade para esta faixa de X, sabendo que esta distribuição é uniforme.
- b) Qual é a probabilidade de a medida da corrente estar entre 5 e 10 miliamperes?
- c) Determine a média (valor esperado) e a variância da distribuição.

Solução:

a) Podemos considerar que:

$X \sim Uniforme(0,10)$

e como $f(x) = 1/(b-a)$, $a \leq x \leq b$, neste caso, $f(x) = 1/((10-0))$, $0 \leq x \leq 10$, ou seja, $f(x) = 0,10$

b) Lembre-se de que a probabilidade de uma variável aleatória contínua é calculada como uma integral (que pode ser calculada pela área sob a curva) para um pequeno intervalo em torno de uma saída específica, e que a fórmula para calcular a área de um retângulo é *base x altura*. Assim, a probabilidade de a medida da corrente estar entre 5 e 10 miliamperes é:

$P(5<x<10) = base\ x\ altura = (10 - 5) \times 0,1 = 5 \times 0,1 = 0,5$

c) Média: $E(x) = (b+a)/2 = (10+0)/2 = 10/2 = 5$

Variância: $V(x) = (b-a)^2/12 = (10-0)^2/12 = 8,333$

Vejamos no R como calcular essa probabilidade, utilizando as funções da biblioteca base `dunif()` e `punif()`:

```
# a) Calculando a função de densidade de probabilidade f(x) = 1/(b-a)
a = 0
b = 10
```

```
print(1/(b-a))

# b) Utilizando a função de distribuição acumulada P(X<=10)-P(X<=5)=
P(5<=X<=10)
print(punif(10,min=0,max=10)-punif(5,min=0,max=10))
# Calculando manualmente
print((10-5)*1/(b-a))

# c) Média e variância
media = (b+a)/2
variancia = (b-a)^2/12
print(cbind(media,variancia))
```

A saída será:

```
[1] 0.1
[1] 0.5
[1] 0.5
     media variancia
[1,]     5  8.333333
```

E o equivalente em Python:

```
# Imports de bibliotecas
from scipy.stats import uniform

# a) Calculando a função de densidade de probabilidade f(x) = 1/(b-a)
a = 0
b = 10
print(1/(b-a))

# b) Utilizando a função de distribuição acumulada P(X<=10)-P(X<=5)=
P(5<=X<=10)
print(uniform.cdf(x=10, loc=0, scale=10) - uniform.cdf(x=5, loc=0,
scale=10))
# Calculando manualmente
print((10-5)*1/(b-a))

# c) Média e variância
media = (b+a)/2
```

```
variancia = (b-a)**2/12
print(media,variancia)
```

A saída será:

```
0.1
0.5
0.5
5.0 8.333333333333334
```

Distribuição Exponencial

Essa distribuição é caracterizada por estar definida para valores positivos, e usualmente é utilizada para representar o tempo. Exemplos: tempo de vida útil de uma lâmpada, tempo de espera em uma fila, tempo de vida útil de equipamentos.

A distribuição exponencial pode ser definida por um parâmetro, **beta** (β), que representa a média da distribuição. Mais formalmente, esta distribuição é definida usando o parâmetro **lambda** (λ), onde β = 1/λ. A distribuição Exponencial é representada pela seguinte fórmula:

$X \sim Exponencial(\lambda)$

$f(x) = \lambda e^{-\lambda x}$, com $x > 0$ e $\lambda > 0$

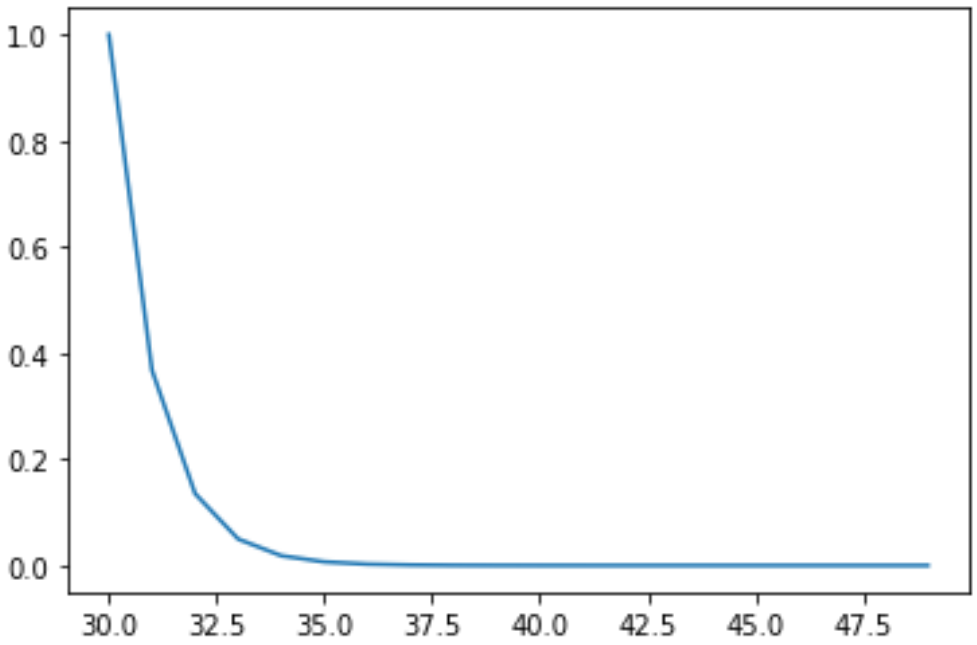
onde:

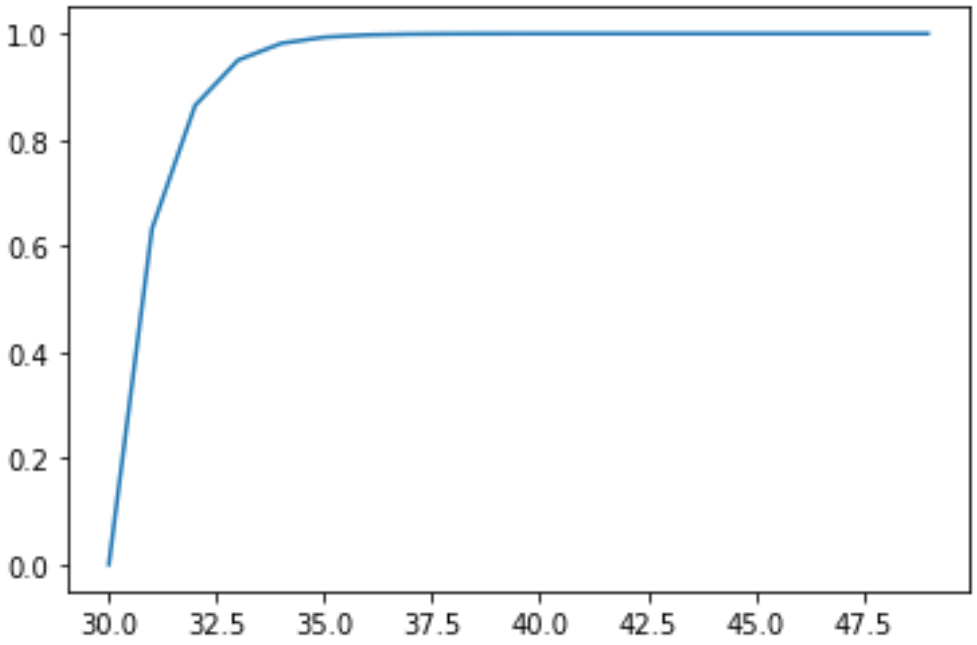
- Valor esperado = $E(x)= 1/\lambda$
- Variância = $V(x) = 1/\lambda^2$

O exemplo de código a seguir, em Python, ilustra o gráfico de uma função de densidade de probabilidade e o gráfico da função de distribuição acumulada de uma distribuição Exponencial com beta = 30 (ou λ 1/30), para um intervalo de observações entre 30 e 50.

```
# Imports de bibliotecas
from scipy.stats import expon
```

```
from matplotlib import pyplot

# Definição dos parâmetros da distribuição
beta = 30

# Criando a distribuição
dist = expon(beta)

# Gráfico da função de densidade de probabilidade
valores = [valor for valor in range(30, 50)]
probabilidades = [dist.pdf(valor) for valor in valores]
pyplot.plot(valores, probabilidades)
pyplot.show()

#  Gráfico da função de distribuição acumulada
acc = [dist.cdf(valor) for valor in valores]
pyplot.plot(valores, acc)
pyplot.show()
```





Vejamos um outro exemplo. A figura a seguir ilustra a Função de Distribuição Acumulada de uma Exponencial:

$$f(x) = \begin{cases} 1 - e^{-\lambda x}, x \geq 0 \\ 0 \qquad\qquad , x < 0 \end{cases}$$

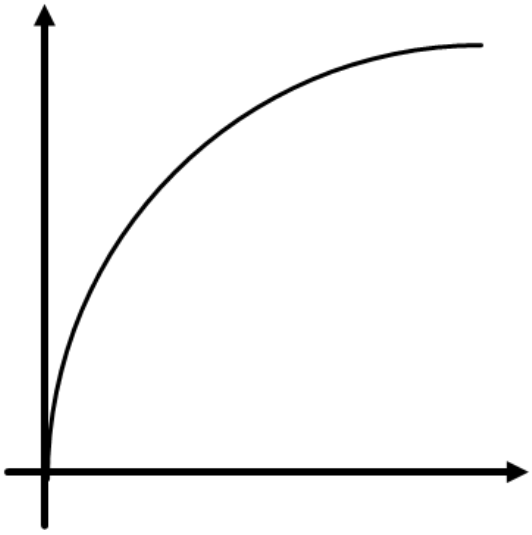


Figura 5.13: Função de Distribuição Acumulada de uma Exponencial

Exemplo

O tempo de espera em uma fila segue uma distribuição exponencial. Se o tempo médio esperado pelo cliente é de 10 minutos para ser atendido, qual a probabilidade:

- a) De que um cliente demore menos do que 12 minutos para ele ser atendido?
- b) De que um cliente demore entre 7 e 12 minutos para ser atendido?

Solução

a) Podemos considerar que: $X \sim Exponencial(\lambda=1/10)$ e como $f(x) = \lambda e^{-\lambda x}$, neste caso, $f(x)=0,1e^{-0,1x}$

Lembre-se de que a probabilidade de uma variável aleatória contínua é calculada como uma integral para um pequeno intervalo em torno de uma saída específica. Então:

Para p(x<12), substituindo na fórmula:

$$f(x < 12) = \int_0^{12} 0,1e^{-0,1x}$$
$$= 1 - e^{-0,1\times 12} = 1 - e^{-1,2}$$

b) Para p(7<x<12):

$$f(7 < x < 12) = f(x < 12) - f(x < 7)$$
$$= (1 - e^{-1,2}) - (1 - e^{-0,7})$$
$$= e^{-0,7} - e^{-1,2}$$

Vejamos no R como calcular essa probabilidade, utilizando as funções da biblioteca base `dexp()` e `pexp()`:

```
# a) Utilizando a função de distribuição acumulada P(X<=12)
print(pexp(12,rate=1/10))

# Calculando manualmente
print(1-exp(-1.2))
```

```
# b) Utilizando a função de distribuição acumulada P(X<=12)-P(X<=7)
print(pexp(12,rate=1/10) - pexp(7,rate=1/10))

# Calculando manualmente
print(1-exp(-1.2) - (1-exp(-0.7)))
```

A saída será:

```
[1] 0.6988058
[1] 0.6988058
[1] 0.1953911
[1] 0.1953911
```

E o equivalente em Python:

```
# Imports de bibliotecas
import math
from scipy.stats import expon

# a) Utilizando a função de distribuição acumulada P(X<=12)
lamb = 1/10
print(expon.cdf(12, loc = 0, scale = 1/lamb))

# Calculando manualmente
print(1-math.exp(-1.2))

# b) Utilizando a função de distribuição acumulada P(X<=12)-P(X<=7)
print(expon.cdf(12,loc = 0, scale = 1/lamb) - expon.cdf(7,scale = 1/lamb))

# Calculando manualmente
print(math.exp(-0.7) - math.exp(-1.2))
```

A saída será:

```
0.6988057880877978
0.6988057880877978
0.19539109187920733
0.1953910918792074
```

Distribuição Normal ou Gaussiana

Essa distribuição é uma das mais conhecidas em estatística e também é chamada de distribuição gaussiana. Exemplos de domínios que podem seguir uma distribuição normal incluem notas de uma turma em uma prova e alturas das pessoas. A distribuição normal tem formato de sino e é uma distribuição simétrica, o que significa que Moda = Média = Mediana. Veja um exemplo a seguir:

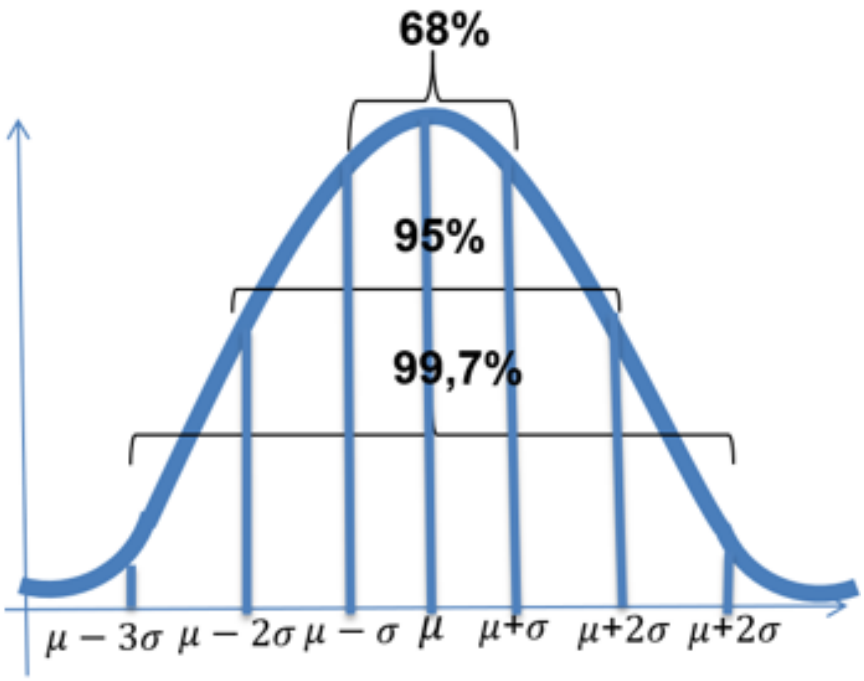


Figura 5.16: Exemplo de Distribuição Normal

Podemos notar na figura anterior que aproximadamente 68% das observações se encontram a um desvio padrão de distância da média, 95% estão a dois desvios da média e 99,7% se encontram a três desvios da média.

A distribuição normal pode ser definida por dois parâmetros, a **média** (μ, o valor esperado) e a **variância** (σ^2, o "quão longe" os valores se encontram da média), e é representada pela seguinte fórmula:

$X \sim Normal(\mu,\sigma^2)$

$$P(X = x) = \frac{\lambda^x e^{-\lambda}}{x!}, x = 0, 1, \ldots; \lambda > 0$$

onde:

- Valor esperado = μ
- Variância = $V(x) = \sigma^2$

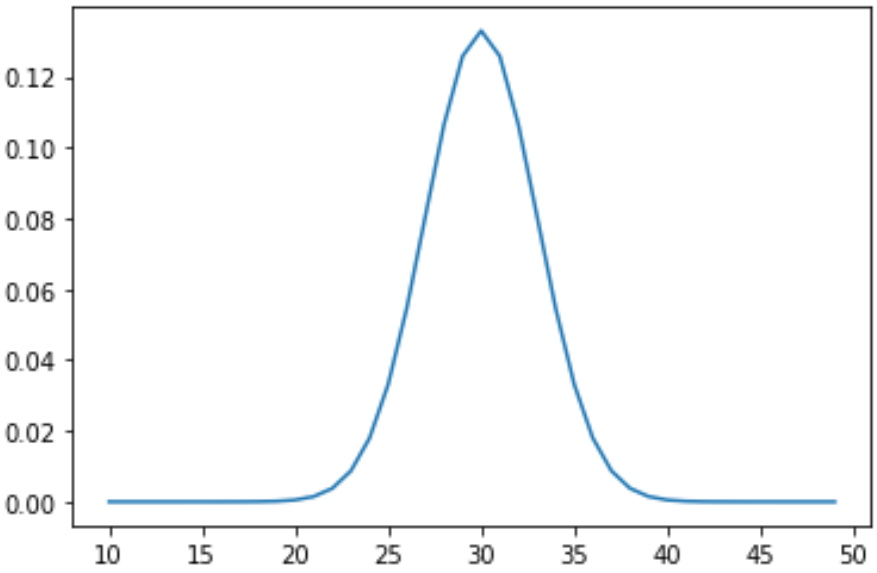
Algumas vezes, no lugar da variância utiliza-se o **desvio padrão** (σ), que pode ser calculado pela raiz quadrada da variância. Uma distribuição normal com média 0 e desvio padrão 1 é chamada **distribuição normal padrão** e, em Ciência de Dados, durante a etapa de análise exploratória, muitas vezes padronizamos nossos dados dessa forma para utilizar uma mesma base de comparação e facilitar a interpretabilidade das análises.

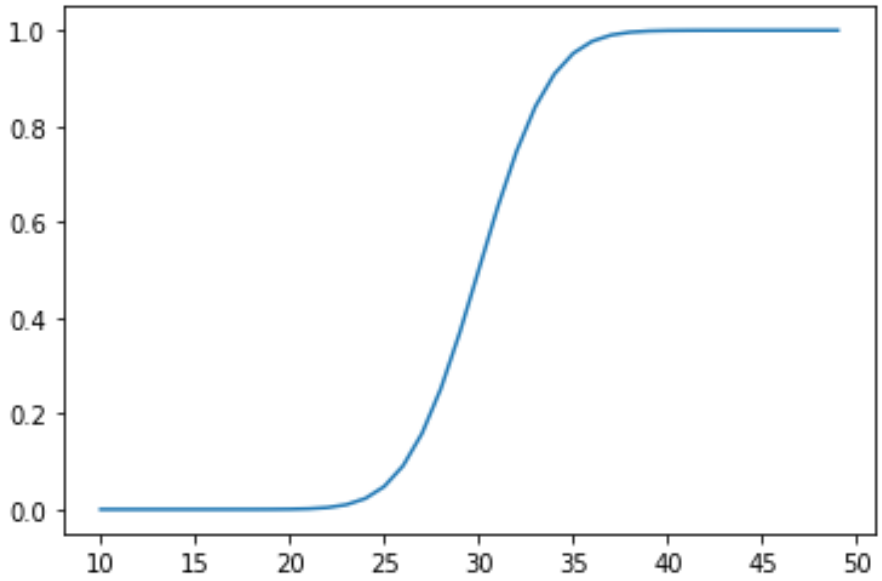
Podemos verificar se uma amostra de dados segue uma distribuição normal através do seu gráfico, ou então usando testes estatísticos (tais como o teste Shapiro-Wilk, que está fora do escopo deste livro). Se as observações de uma variável aleatória são normalmente distribuídas, isso significa que elas podem ser resumidas apenas pela sua média e variância (ou desvio padrão). Podemos calcular a probabilidade de cada observação usando a função de densidade de probabilidade e, se o gráfico tiver o formato de um sino, confirmaremos que estes dados seguem uma distribuição normal.

O exemplo de código a seguir, em Python, ilustra o gráfico de uma função de densidade de probabilidade e o gráfico da função de distribuição acumulada de uma distribuição Normal com média 30 e desvio padrão 3. Repare que a probabilidade de um valor igual à média é em torno de 12%.

```
# Imports de bibliotecas
from scipy.stats import norm
from matplotlib import pyplot

# Definição dos parâmetros da distribuição
```

```
media = 30
desvio_padrao = 3

# Criando a distribuição
dist = norm(media, desvio_padrao)

# Gráfico da função de densidade de probabilidade
valores = [valor for valor in range(10, 50)]
probabilities = [dist.pdf(valor) for valor in valores]
pyplot.plot(valores, probabilities)
pyplot.show()

#  Gráfico da função de distribuição acumulada
acc = [dist.cdf(valor) for valor in valores]
pyplot.plot(valores, acc)
pyplot.show()
```





Cálculo de probabilidades em distribuições normais

Não é tão simples efetuar o cálculo da probabilidade em variáveis que seguem distribuição normal como nas outras distribuições, uma vez que a normal é uma distribuição contínua não integrável analiticamente. Assim, somente conseguimos calcular probabilidades por meio de métodos numéricos iterativos, não possuindo solução analítica para este problema. Para tal, fazemos uso de uma **Tabela Normal**, que fornece as probabilidades associadas aos quantis de uma variável aleatória padronizada (Z), que possui uma distribuição normal padrão (com média 0 e desvio padrão 1). Z pode ser calculada como:

$Z = (X-\mu)/\sigma$

Existem algumas tabelas da distribuição normal, todas elas representam valores de probabilidade de uma distribuição normal padrão (média 0 e variância 1). Há duas formas de interpretação:

1) Observamos os quantis da distribuição normal, que são representados em sua parte inteira e primeira decimal pelas linhas e seu segundo decimal nas colunas. Por exemplo, se queremos encontrar a probabilidade de ocorrer um valor entre 0 e 1,64,

olharemos nas linhas qual valor que corresponde a (1,6) e na coluna olharemos a segunda casa decimal (4). Ao encontrá-los, em seguida, cruzaremos a informação de linha e coluna e veremos a probabilidade no miolo da tabela, que nesse caso corresponde a 0,44950. Ou seja, a probabilidade de se observar em uma distribuição normal padrão um valor de 0 a 1,64 é 0,449 aproximadamente ou 44,9%.

2) A segunda forma é olhando pela ótica inversa: agora não mais queremos a probabilidade de esta assumir o valor em um intervalo, queremos o quantil da normal que representa essa probabilidade. Por exemplo, queremos encontrar o quantil da normal que representa a probabilidade de 0,449. Para tal, basta olhar o miolo da tabela e cruzar linha e coluna a fim de observar qual valor que se forma ao unir a linha (1,6) com a coluna (4) e chegaremos à conclusão de que o quantil da normal padrão que representa a probabilidade de 0,449 será o 1,64.

Exemplo

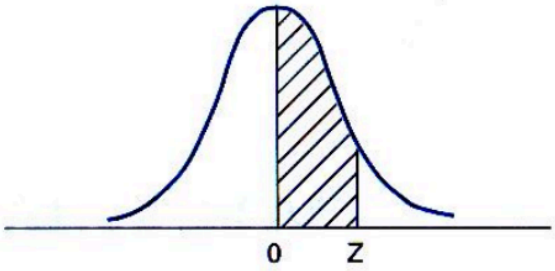
Considere que o peso dos alunos segue uma normal com média igual a 70 kg e desvio padrão igual a 5 kg. Calcule a probabilidade de um aluno possuir um peso entre 60 e 80 kg.

Solução

Note que não podemos resolver este problema de imediato, pois não sabemos nada acerca da distribuição normal com média 70 kg e desvio padrão 5 kg. Para a normal padrão, com média 0 e desvio padrão (ou variância) 1, temos os valores das probabilidades associadas, então basta padronizarmos a variável *x*. Como queremos a probabilidade de o aluno ter um peso entre 60 kg e 80 kg, ou seja, $P(60 < x < 80)$, faremos a padronização com auxílio das tabelas da distribuição normal.

$$P\left(\frac{(60-70)}{5} < x < \frac{(80-70)}{5}\right) = P(-2 < Z < 2)$$

$$2P(0 < Z < 2) = 2(0,47725) = 0,9545\ ou\ 95,45\%.$$



z	0	1	2	3	4	5	6	7	8	9
0,0	0,0000	0,0040	0,0080	0,0120	0,0160	0,0199	0,0239	0,0279	0,0319	0,0359
0,1	0398	0438	0478	0517	0557	0596	0636	0675	0714	0754
0,2	0793	0832	0871	0910	0948	0987	1026	1064	1103	1141
0,3	1179	1217	1255	1293	1331	1368	1406	1443	1480	1517
0,4	1554	1591	1628	1664	1700	1736	1772	1808	1844	1879
0,5	1915	1950	1985	2019	2054	2088	2123	2157	2190	2224
0,6	2258	2291	2324	2357	2389	2422	2454	2486	2518	2549
0,7	2580	2612	2642	2673	2704	2734	2764	2794	2823	2852
0,8	2881	2910	2939	2967	2996	3023	3051	3078	3106	3133
0,9	3159	3186	3212	3238	3264	3289	3315	3340	3365	3389
1,0	3413	3438	3461	3485	3508	3531	3554	3577	3599	3621
1,1	3643	3665	3686	3708	3729	3749	3770	3790	3810	3830
1,2	3849	3869	3888	3907	3925	3944	3962	3980	3997	4015
1,3	4032	4049	4066	4082	4099	4115	4131	4147	4162	4177
1,4	4192	4207	4222	4236	4251	4265	4279	4292	4306	4319
1,5	4332	4345	4357	4370	4382	4394	4406	4418	4429	4441
1,6	4452	4463	4474	4484	4495	4505	4515	4525	4535	4545
1,7	4554	4564	4573	4582	4591	4599	4608	4616	4625	4633
1,8	4641	4649	4656	4664	4671	4678	4686	4693	4699	4706
1,9	4713	4719	4726	4732	4738	4744	4750	4756	4761	4767
2,0	4772	4778	4783	4788	4793	4798	4803	4808	4812	4817
2,1	4821	4826	4830	4834	4838	4842	4846	4850	4854	4857

Vejamos no R como calcular essa probabilidade, utilizando as funções da biblioteca base `dnorm()` e `pnorm()`:

```
# Utilizando a função de distribuição acumulada P(x<=80) - P(x<=60)
print(pnorm(80, mean = 70, sd = 5) - pnorm(60, mean = 70, sd = 5))

# Calculando manualmente
limite_inf = (60-70)/5
limite_sup = (80-70)/5

# Utilizando a função de distribuição acumulada da normal padrão P(z<=2) - P(z<=-2)
print(pnorm(limite_sup) - pnorm(limite_inf))
```

A saída será:

```
[1] 0.9544997
[1] 0.9544997
```

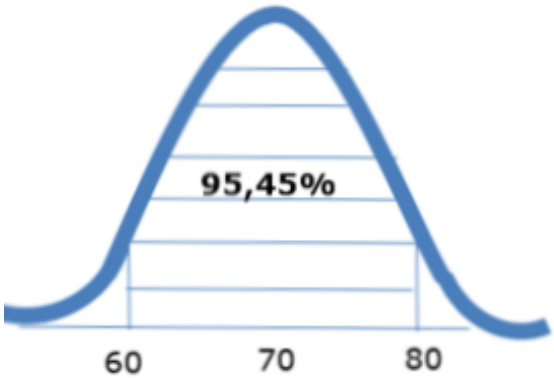
E o equivalente em Python:

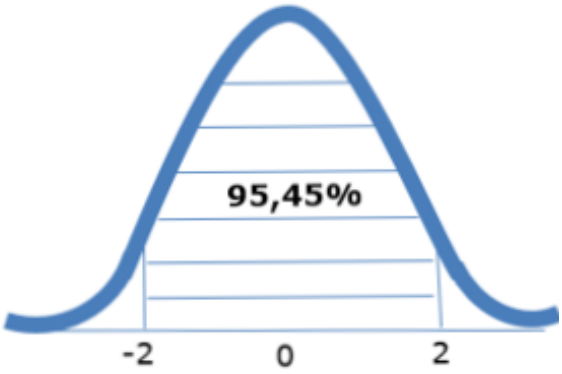
```
# Imports de bibliotecas
from scipy.stats import norm

# Utilizando a função de distribuição acumulada P(x<=80) - P(x<=60)
print(norm.cdf(x=80, loc=70, scale=5) - norm.cdf(x=60, loc=70, scale=5))

# Calculando manualmente
limite_inf = (60-70)/5
limite_sup = (80-70)/5

# Utilizando a função de distribuição acumulada da normal padrão P(z<=2) - P(z<=-2)
print(norm.cdf(x=limite_sup) - norm.cdf(x=limite_inf))
```

A saída será:

```
0.9544997361036416
0.9544997361036416
```

Entendendo graficamente:

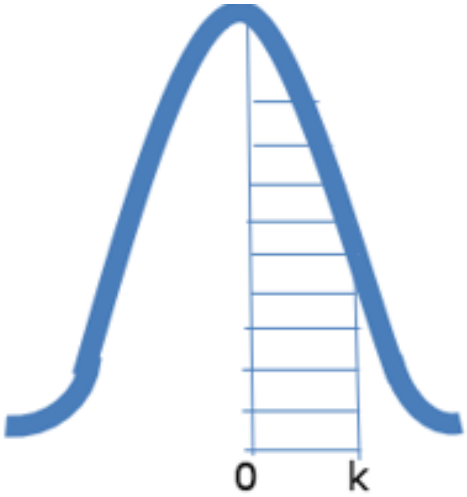




A distribuição normal tem uma heurística que define a porcentagem de dados cobertos por um determinado intervalo usando o número de desvios padrão da média. Esta heurística é conhecida como *regra empírica*, que representa a porcentagem aproximada dos dados cobertos pelos intervalos definidos respectivamente por 1, 2 e 3 desvios padrão da média. Assim, esperamos que 68% dos dados sejam cobertos por valores equidistantes a 1 desvio padrão da média acima ou abaixo da mesma, que 95% dos dados sejam cobertos por valores equidistantes a 2 desvios padrão da média acima ou abaixo da mesma e que 99,7% dos dados sejam cobertos por valores equidistantes a 3 desvios padrão da média acima ou abaixo dela.

Observação: existem Quantis da Normal que serão úteis nos capítulos de intervalos de confiança e testes de hipóteses, portanto, é importante estudá-los. São eles:

- $z_{0,05}$ = Quantil de k tal que P(0<Z<K) = 45%; K = 1,64.
- $z_{0,025}$ = Quantil de k tal que P(0<Z<K) = 47,5%; K = 1,96.
- $z_{0,005}$ = Quantil de k tal que P(0<Z<K) = 49,5%; K = 2,58.



Em Ciência de Dados, é útil conhecer a função de densidade de probabilidade para uma amostra de dados para verificar, por exemplo, se uma determinada observação é tão improvável a ponto de ser considerada um *outlier* e precisar ser removida da amostra. Além disso, a distribuição de probabilidade dos dados de entrada pode ajudar a escolher algoritmos de Machine Learning mais apropriados. Na prática, é muito difícil conhecer exatamente função de densidade de probabilidade para uma amostra aleatória de dados, mas ela pode ser aproximada.

Neste capítulo, falamos das distribuições de probabilidade discretas e contínuas. Vimos que podemos resumir a probabilidade de resultados para variáveis aleatórias discretas usando distribuições de probabilidade discretas. Quando estamos interessados em um único resultado binário, podemos utilizar a distribuição de Bernoulli e quando estamos interessados em uma sequência de resultados binários, usamos a distribuição Binomial. Já quando estamos interessados no número de ocorrências de um evento em um intervalo correspondente, podemos usar a distribuição de Poisson. Vimos também que podemos resumir a probabilidade de resultados para variáveis aleatórias contínuas usando distribuições de probabilidade contínuas, sendo exemplos deste tipo de distribuição Uniforme, Exponencial e Normal.

Capítulo 6
Inferência Estatística: Noções de amostragem e reamostragem

Conforme vimos nos capítulos anteriores, a Estatística pode ser subdivida em três grandes áreas: Estatística Descritiva, Probabilidade e Inferência Estatística. Neste capítulo, serão abordados assuntos relacionados à Inferência Estatística, que é o estudo de técnicas que possibilitam a extrapolação das informações e conclusões obtidas a partir da amostra para um grande conjunto de dados.

Neste contexto, **amostragem** é o conjunto de técnicas para selecionar uma amostra da população, com o objetivo de obter informações de uma ou mais características de interesse, as quais permitam chegar a conclusões a respeito dos parâmetros desta população.

Conforme já vimos no capítulo 2, **população** (N) é a coleção de todos os indivíduos que possuem determinadas características, as quais estamos interessados em estudar; **amostra** (n) é um subconjunto da população, uma parte dos indivíduos que possuem determinadas características, e **censo** é o exame de todos os elementos da população. Vamos introduzir dois novos conceitos:

- **Parâmetro-alvo:** quantidade desconhecida, na qual temos interesse.
- **Erro amostral:** é a diferença entre um resultado amostral e o verdadeiro resultado populacional.

Vamos falar um pouco mais do erro amostral. Já sabemos que por meio da amostragem podemos realizar uma Inferência Estatística sobre a população a partir de um pequeno conjunto de observações (amostra), podendo então generalizar propriedades da amostra para a população. O processo de trabalhar com a amostra é muito mais rápido do que trabalhar com toda a população, mas ele certamente

conterá erros. Os principais de erros incluem viés de seleção e erro de amostragem, a saber:

- **Viés de seleção:** quando o método de extrair observações distorce a amostra de alguma maneira.
- **Erro de amostragem:** causado devido à natureza aleatória do processo de amostragem.

Sempre que possível, devemos analisar a população através de estatísticas resumidas e visualizações, a fim de encontrar eventuais erros no processo de amostragem e verificar se as propriedades da amostra são compatíveis com as da população.

Exemplo:

Imagine que queiramos estudar a altura média dos alunos em uma sala de aula com 300 alunos. Neste caso, temos:

- **População:** todos os alunos da sala de aula.
- **Amostra:** alunos selecionados da população por uma técnica de amostragem.
- **Parâmetro-alvo:** altura média dos alunos na população.

Em Ciência de Dados, geralmente utilizamos amostras para medir algo (utilizando estatística amostral) ou modelar algo (como um modelo de Machine Learning). Dependendo da amostra selecionada, os resultados obtidos podem ser diferentes. Por exemplo, quando muitos dados estão disponíveis, é possível extrair diversas amostras, treinar um modelo de Machine Learning com cada uma delas e observar diretamente a distribuição de uma estatística amostral, como a sua média, para estimar o desempenho deste modelo. Quando não temos muitos dados, uma opção é treinar um modelo usando a maior quantidade de dados disponível.

É importante diferenciar a distribuição dos dados (que consiste na distribuição de pontos de dados individuais do *dataset*) da distribuição amostral (que consiste na distribuição de uma estatística amostral, como a média). Em geral, a distribuição de uma estatística

amostral é mais regular, mais estreita e mais próxima do formato de sino do que a distribuição dos próprios dados, e quanto maior a amostra em que a estatística se baseia, mais isso é verdade.

Para melhor entendimento deste conceito, observe os gráficos a seguir. O primeiro corresponde ao histograma de uma amostra aleatória de 500 números inteiros (que denominaremos *dataset* original); o segundo, a uma amostra de 500 médias de 5 valores sorteados do *dataset* original e o terceiro, a uma amostra de 500 médias de 20 valores sorteados do *dataset* original. Repare que o histograma dos valores individuais do *dataset* original é mais espalhado, enquanto o histograma das médias de 5 elementos é mais compacto e se aproxima de uma distribuição normal, e o histograma das médias de 20 elementos é ainda mais compacto e mais próximo de uma distribuição normal.

Observação: é possível encontrar o código para gerar estes gráficos no repositório do GitHub do livro, porém, observe que os resultados podem variar devido à aleatoriedade do processo de amostragem.

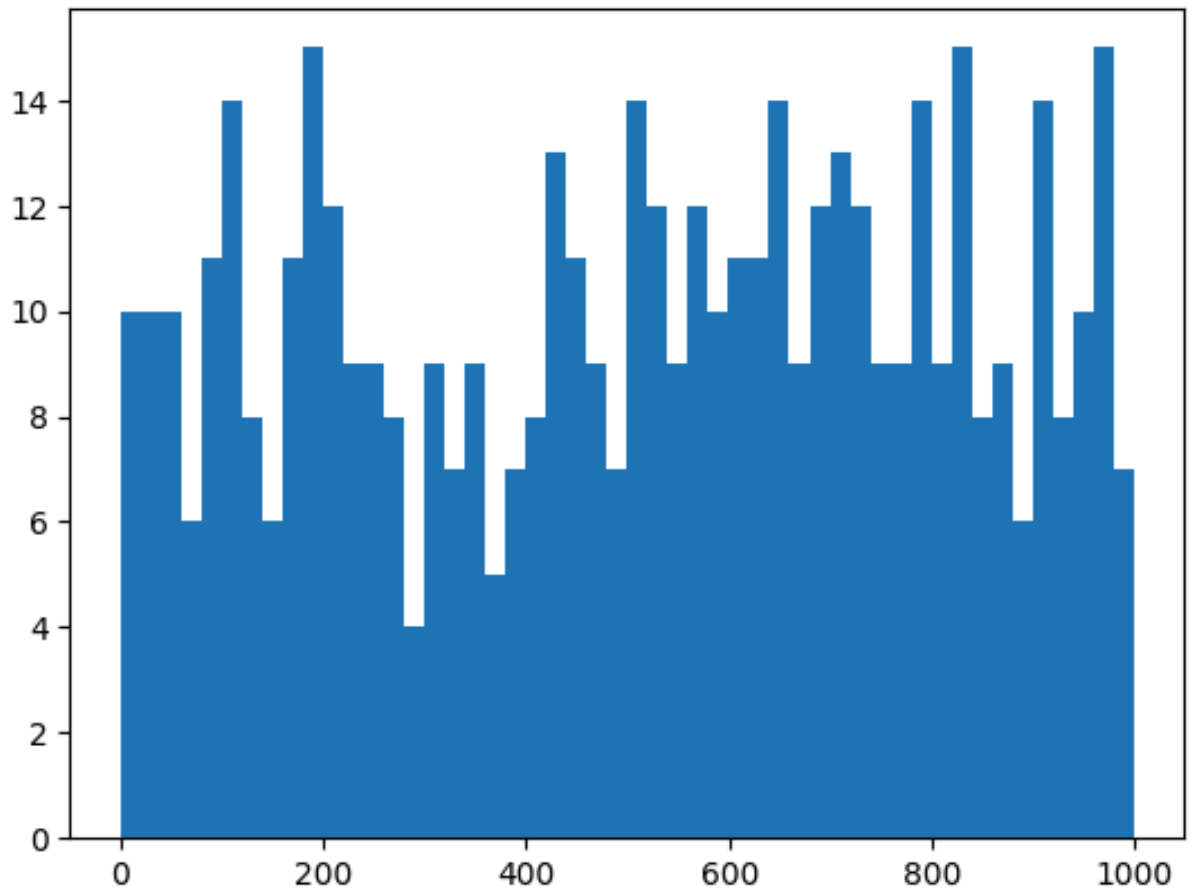


Figura 6.1: Histograma de 500 elementos.

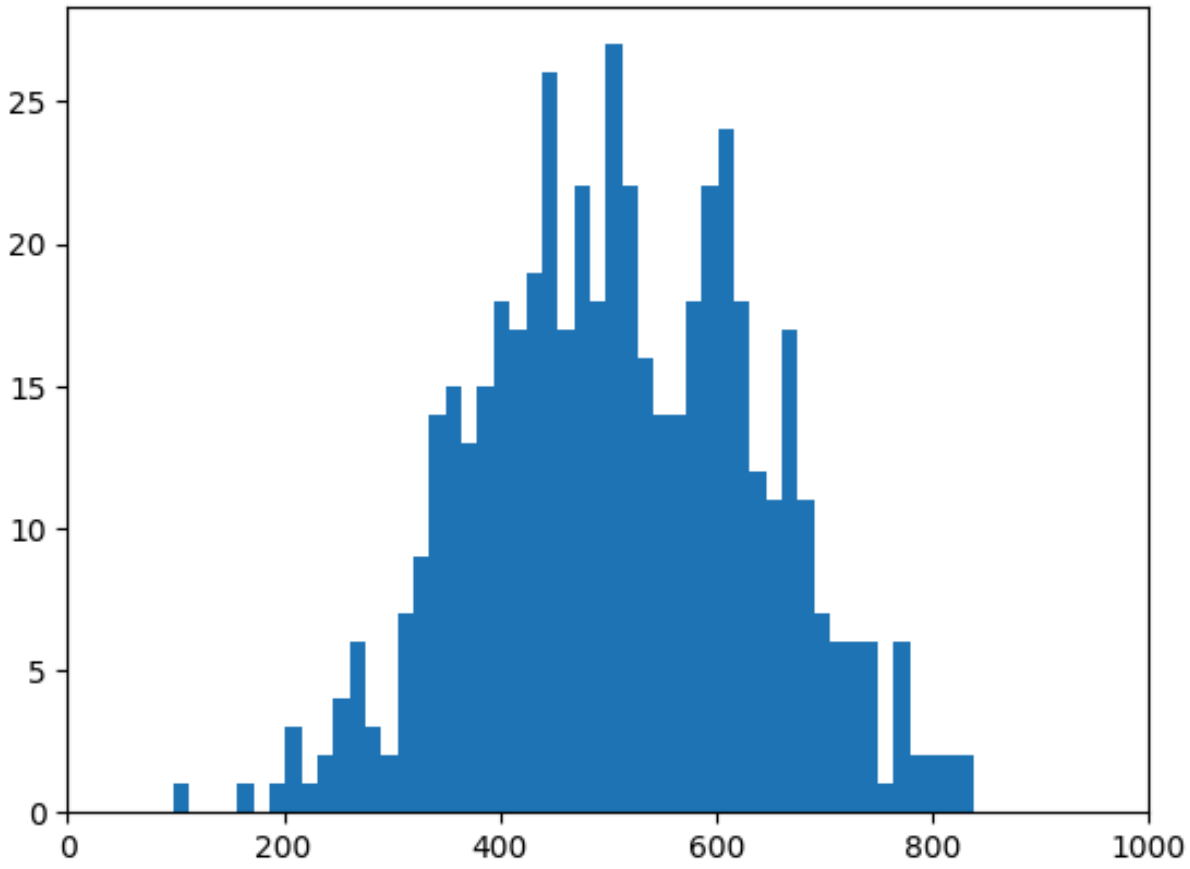


Figura 6.2: Histograma de 500 médias de 5 elementos.

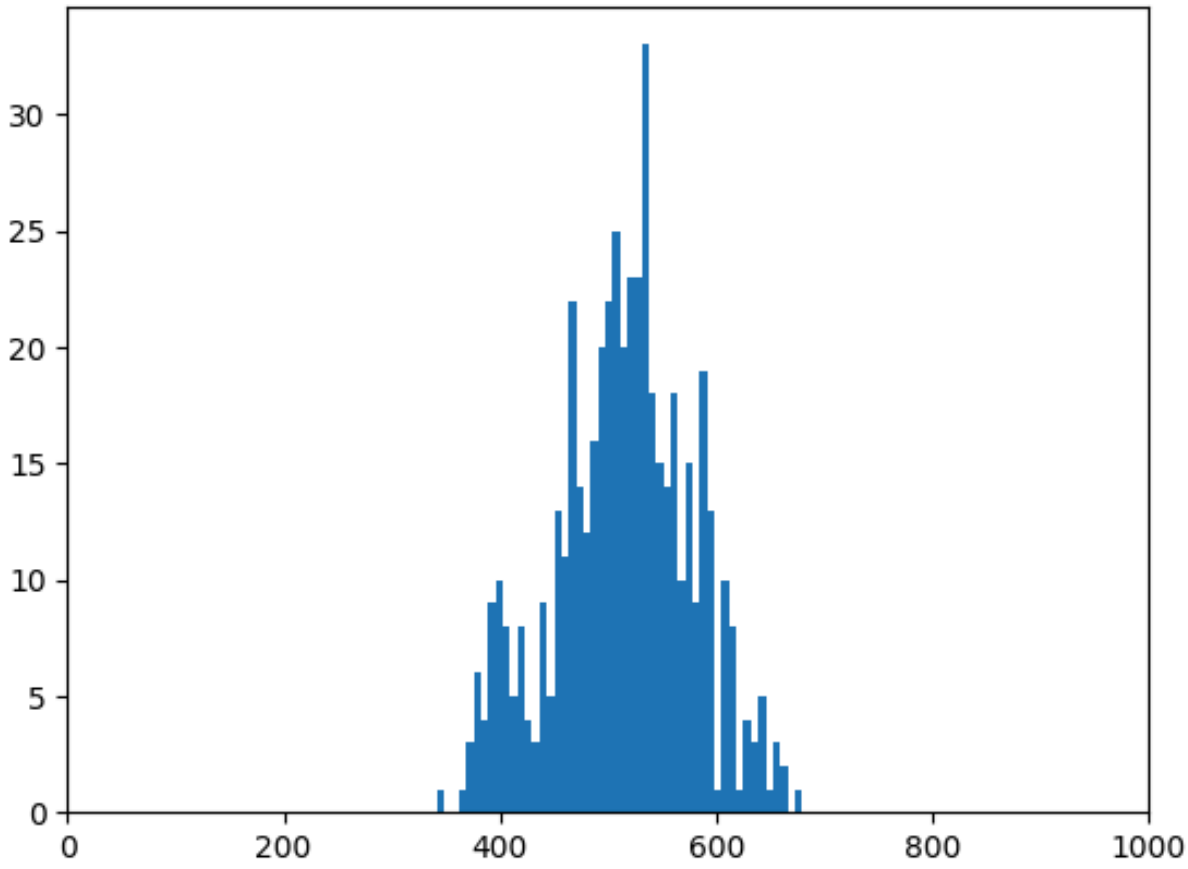


Figura 6.3: Histograma de 500 médias de 20 elementos.

Esse conceito consiste no **Teorema central do limite**, que será definido formalmente a seguir.

6.1 Teorema central do limite

Também chamado de teorema do limite central, é um dos teoremas mais importantes e fortes da Estatística e afirma que, quando a amostra aumenta significativamente ($n > 30$), a distribuição amostral da soma e da média de variáveis aleatórias independentes convergem para uma distribuição normal, ou seja, $Z \sim N(0,1)$. Isso

ocorre para quaisquer variáveis, independente de a população origem ser normalmente distribuída ou padronizada. Na Inferência Estatística, este teorema é útil, por exemplo, para estimar parâmetros como a média populacional ou o desvio padrão da média populacional a partir de uma amostra aleatória dessa população.

Ou seja:

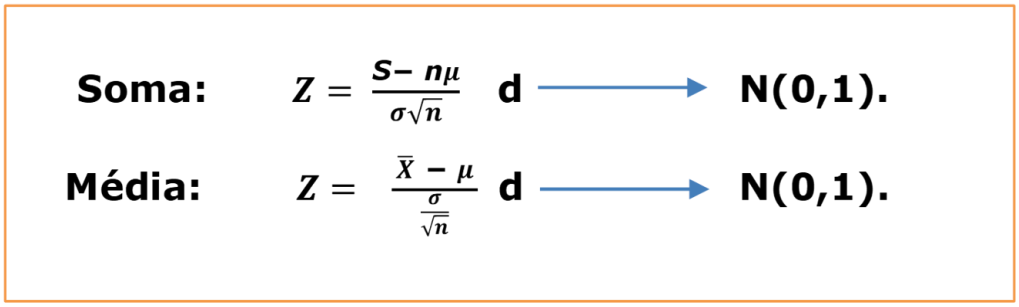


Figura 6.4: Teorema central do limite.

Onde:

- $\overline{X}$ é a média amostral
- **μ** é média populacional
- **σ** é a variância populacional
- **S** é a soma amostral
- **n** é o tamanho amostral
- **d** → é a notação de "convergência em distribuição para"

A figura a seguir ilustra o teorema central do limite na soma de dados. Na primeira vez, jogamos 30 vezes somente um dado e observamos as repetições de cada face, já na segunda observamos a soma de dois dados jogados também 30 vezes, e assim repetimos o experimento, até chegarmos à soma das faces de seis dados jogados 30 vezes. Note que, à medida que aumentamos a quantidade de dados a serem jogados e verificamos a distribuição da

soma, mais nos aproximamos ao formato de uma distribuição de sino, ou seja, de uma distribuição normal de probabilidade.

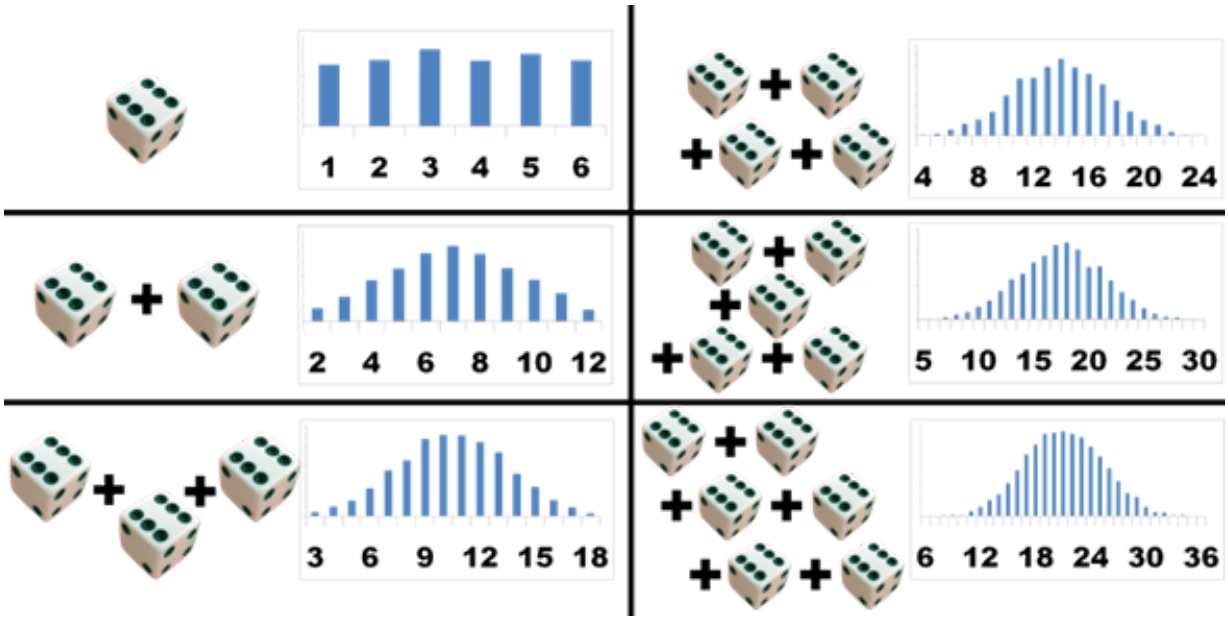


Figura 6.6: Exemplo do aplicação do teorema central do limite.

A interpretação empírica do gráfico desta distribuição é que quando nos distanciamos da média, a probabilidade de ocorrência é menor. Ou seja, é mais provável que ocorra um evento em torno da média do que distante dela.

Em Ciência de Dados, o teorema central do limite é utilizado, por exemplo, para realizar inferência sobre médias amostrais em relação as populacionais e para executar testes de hipóteses.

As amostragens podem ser do tipo *Casual/Probabilística* ou *Não casual/Não probabilística*. As técnicas de amostragem Probabilísticas são mais vantajosas por possibilitarem o cálculo de estimativas de precisão e margens de erro, possibilitando a expansão dos resultados amostrais obtidos para os populacionais. Vamos estudar cada um destes tipos a seguir.

6.2 Tipos de amostragem casuais/probabilísticas

Nas **amostragens casuais/probabilísticas**, todos os elementos possuem uma probabilidade de inclusão conhecida e não nula e as probabilidades de seleção das amostras são possíveis de se calcular. São exemplos a Amostragem Aleatória Simples, a Amostragem Sistemática, a Amostragem Estratificada e a Amostragem por Conglomerados, detalhadas nas próximas subseções e ilustradas pela figura a seguir:

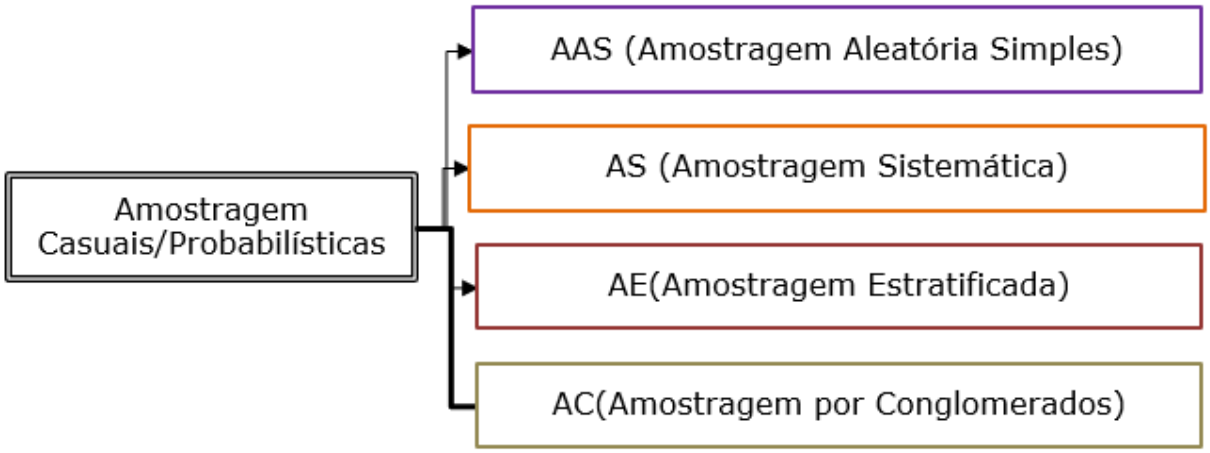


Figura 6.7: Tipos de Amostragem Casuais/Probabilísticas

Amostragem Aleatória Simples (AAS)

Consiste em uma técnica de amostragem em que todos os elementos possuem equiprobabilidade, ou seja, a mesma probabilidade de serem incluídos na amostra. É uma das técnicas mais simples de amostragem. A AAS Pode se dar de duas formas:

- **Com reposição:** O elemento que for retirado é recolocado novamente para sorteio, não alterando o espaço amostral. Exemplo: colocar o nome de várias pessoas em uma sacola e realizar um sorteio com o nome das pessoas. Uma vez que é

retirado um nome, este volta para o saco para um eventual novo sorteio.

- **Sem reposição:** o elemento que for retirado não é recolocado novamente para sorteio, alterando o espaço amostral. Exemplo: colocar o nome de várias pessoas em uma sacola e realizar um sorteio com o nome das pessoas. Uma vez que é retirado um nome, este não volta para o saco para eventual novo sorteio.

Trabalhando no R com amostragem com e sem reposição, a função `sample()` do pacote base do R retira uma amostra aleatória simples com ou sem reposição, bastando utilizar o argumento *replace=T* e *replace=F*, respectivamente. Digite o trecho de código a seguir e veja o resultado:

```
# Fixando a semente aleatória para garantir a reprodutibilidade dos resultados
set.seed(7)

# População a ser amostrada
populacao = c(1,2,3,4,5,6,7,8,9,10)

# AAS com reposição de tamanho 5
print(sample(populacao, size=5, replace=T))

# AAS sem reposição de tamanho 7
print(sample(populacao, size=7, replace=F))
```

A saída resultante será:

```
[1] 10  3  7  2 10
[1]  6  8  9  3 10  7  2
```

O código equivalente em Python seria o seguinte, usando a função `resample` do pacote *sklearn.utils*:

```
# Imports de bibliotecas
from sklearn.utils import resample

# População a ser amostrada
```

```
populacao = [1,2,3,4,5,6,7,8,9,10]

# AAS com reposição de tamanho 5
print(resample(populacao, replace=True, n_samples=5, random_state=7))

# AAS sem reposição de tamanho 7
print(resample(populacao, replace=False, n_samples=7, random_state=7))
```

A saída resultante será:

```
[5, 10, 7, 4, 4]
[9, 6, 1, 3, 2, 10, 8]
```

Amostragem Sistemática (AS)

Consiste em um caso particular de amostragem aleatória simples em que os elementos estão ordenados. O processo de seleção consiste em escolher um número *r*, entre 1 e *k* (intervalo de seleção), sendo *k = N/n*. Este *r* será denominado como **partida aleatória**. O primeiro elemento será aquele que, ordenado, possuir a r-ésima posição; o segundo pertencente à amostra será o de posição *r+k*, o terceiro, *r+2k*, e assim por diante (*r*, *r+k*, *r+2k* ... *r+(n-1)k*).

Exemplo:

Queremos escolher uma amostra de 5 fichas, dentre uma população de 5000 fichas. Temos:

- N = 5000 e n = 5
- k = N/n = 5000/5 = 1000

Sortearemos um número entre 1 e 1000 e o número sorteado será a posição da primeira ficha. Vamos supor que sorteamos o número 298, portanto a primeira ficha a pertencer à amostra é a que está na posição 298. A segunda ficha a pertencer à amostra será a ficha de posição 298 + 1000 = 1298, ou seja, a ficha de número 1298.

Vamos repetir esse procedimento até a ficha de posição 298 + 1000 x (5 - 1), logo a última ficha terá posição 4298.

Vejamos este exemplo no R:

```
# Fixando a semente aleatória para garantir a reprodutibilidade dos
resultados
set.seed(7)

# Queremos escolher uma amostra (n) de 5 fichas, dentre uma população (N)
de 5000 fichas:                -
n = 5
N = 5000

# Calculando o valor de k
k = floor(N / n)  # 1000 (divisão inteira)

# Sorteio da partida aleatória entre 1 e k
r = sample(1:k, size = 1)

# Gerando a amostra sistemática
amostra = seq(r, N, by = k)

# Exibindo a amostra
print(amostra)
```

A saída resultante será:

```
[1]    298 1298 2298 3298 4298
```

O código equivalente em Python seria o seguinte, usando a biblioteca `resample` do pacote *sklearn.utils* (note que os valores resultantes são diferentes do exemplo em R devido à aleatoriedade):

```
# Imports de bibliotecas
import random

# Fixando a semente aleatória para garantir a reprodutibilidade dos
```

```
resultados
random.seed(7)

# Queremos escolher uma amostra (n) de 5 fichas, dentre uma população (N)
de 5000 fichas:                -
n = 5
N = 5000

# Calculando o valor de k
k = N // n  # 1000 (divisão inteira)

# Sorteio da partida aleatória entre 1 e k
r = random.randint(1, k)

# Gerando a amostra sistemática
amostra = list(range(r, N + 1, k))

# Exibindo a amostra
print(amostra)
```

A saída resultante será:

```
[332, 1332, 2332, 3332, 4332]
```

Amostragem Estratificada (AE)

É uma técnica de amostragem utilizada quando estamos trabalhando com populações heterogêneas. Consiste em inicialmente dividir a população em estratos (subconjuntos), de forma que os elementos de um conjunto sejam homogêneos entre si e heterogêneos quando comparados com elementos de outros subconjuntos. Em seguida, extraímos de cada subconjunto uma amostra independente com tamanhos prefixados, com o objetivo de melhorar a eficiência. Esse processo em geral faz com que se aumente a representatividade da amostra.

Exemplo:

Queremos extrair uma amostra de tamanho n = 60 de uma população de tamanho N = 2000, que consiste de 4 estratos de tamanhos N1 = 500, N2 = 1200, N3 = 200 e N4 = 100, representando 4 cidades diferentes. Se a alocação deve ser proporcional, qual o tamanho da amostra a ser extraída de cada um dos quatro estratos?

Vamos resolver exemplo no R:

```
# Tamanho da amostra (n) e dos estratos de população (N1 a N4)
n = 80
N1 = 500
N2 = 1200
N3 = 200
N4 = 100

# Tamanho total da população
N = N1 + N2 + N3 + N4

# Fração amostral
f = n / N

# Calculando os n's para cada extrato com alocação proporcional ao tamanho
do estrato
n1 = f*N1
n2 = f*N2
n3 = f*N3
n4 = f*N4

# Exibindo os tamanhos das amostras dos estratos
cbind(n1, n2, n3, n4)
```

A saída resultante será:

```
     n1 n2 n3 n4
[1,] 20 48  8  4
```

O código equivalente em Python é bem parecido:

```
# Tamanho da amostra (n) e dos estratos de população (N1 a N4)
n = 80
N1 = 500
N2 = 1200
N3 = 200
N4 = 100

# Tamanho total da população
N = N1 + N2 + N3 + N4

# Fração amostral
f = n / N

# Calculando os n´s para cada extrato com alocação proporcional ao tamanho
do estrato
n1 = f*N1
n2 = f*N2
n3 = f*N3
n4 = f*N4

# Exibindo os tamanhos das amostras dos estratos
print("n1=%d n2=%d n3=%d n4=%d" % (n1, n2, n3, n4))
```

A saída resultante será:

```
n1=20 n2=48 n3=8 n4=4
```

Em Ciência de Dados, utilizamos a Amostragem Estratificada quando trabalhamos com problemas de aprendizado supervisionado em Machine Learning e fazemos a divisão de um *dataset* em conjuntos de treino e teste, mas queremos garantir que a separação seja estratificada por alguma característica, por exemplo, a classe em problemas de classificação.

Vejamos um exemplo em Python desta operação usando o *dataset* Breast Cancer, que contém 212 linhas da classe 0 (maligno) e 357 linhas da classe 1 (benigno). Usaremos a função `train_test_split` da biblioteca *Scikit-Learn*, configurando o parâmetro `stratify` para que a divisão dos conjuntos seja estratificada:

```
# Imports de bibliotecas
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

# Carregando o dataset Breast Cancer, convertendo para DataFrame e
separando em X (atributos) e y (classe)
dataset = load_breast_cancer()
X = pd.DataFrame(dataset.data)
y = pd.DataFrame(dataset.target)

# Dividindo o dataset em conjuntos de treino (80%) e teste (20%),
estratificando por y (classe do problema)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
stratify=y, random_state = 7)

# Exibindo o tamanho de cada conjunto
print("X: {}".format(X.shape) + " - Y: {}".format(y.shape))
print("X_train: {}".format(X_train.shape) + " - y_train:
{}".format(y_train.shape))
print("X_test: {}".format(X_test.shape) + " - y_test:
{}".format(y_test.shape))
```

A saída resultante será:

```
X: (569, 30) - Y: (569, 1)
X_train: (455, 30) - y_train: (455, 1)
X_test: (114, 30)- y_test: (114, 1)
```

Amostragem por Conglomerado ou Cluster (AC)

É um método bastante utilizado por praticidade e economicidade. Consiste em selecionar grupos de unidades (conglomerados) e não unidades de análise (indivíduos da população). Em geral, os conglomerados são escolhidos de forma aleatória, de forma a serem heterogêneos dentro de um mesmo conglomerado, para que representem a diversidade da população. Em seguida, são analisados todos os elementos que os compõem.

Exemplo:

Estamos interessados em estudar o peso dos alunos de algumas escolas no RJ. Temos então que os conglomerados a serem sorteados serão as escolas, e dentro das escolas, vamos entrevistar todos os alunos de todas as turmas.

Vejamos um exemplo em Python, extraindo um conglomerado de tamanho 50 no *dataset Breast Cancer*:

```
import pandas as pd
from sklearn.datasets import load_breast_cancer

# Carregando o dataset Breast Cancer, convertendo para DataFrame e
adicionando a variável target
dataset = load_breast_cancer()
df = pd.DataFrame(dataset.data, columns=dataset.feature_names)
df['target'] = dataset.target

# Dimensões do dataset original
print(df.shape)

# Amostragem por conglomerado - a amostra terá tamanho 50
amostra = df.sample(50, random_state=7)

# Dimensões da amostra
print(amostra.shape)
```

A saída resultante será:

```
(569, 31)
(50, 31)
```

Resumo das amostragens probabilísticas

Amostragem Aleatória Simples (AAS)

- Vantagens: simplicidade; estimativas não viciadas.
- Desvantagens: requer cadastro prévio; custo elevado.

Amostragem Sistemática (AS)

- Vantagens: agilidade na execução; mais eficiente que AAS quando a população é homogênea.
- Desvantagens: requer cadastro prévio; menos eficiente que AAS quando a população não é homogênea.

Amostragem Estratificada (AE)

- Vantagens: menor custo que AAS; espalhamento da Amostra.
- Desvantagens: maior eficiência que AAS, em geral; se os estratos forem heterogêneos dentro, apresenta uma menor eficiência.

Amostragem por Conglomerados (AC)

- Vantagens: praticidade; economicidade.
- Desvantagens: menos eficiente que outros métodos.

6.3 Tipos de amostragem não casuais/não probabilísticas

Em geral, as **amostragens não casuais/não probabilísticas** possuem um teor subjetivo na escolha dos elementos amostrais, dependendo de critérios e julgamentos do pesquisador. São exemplos a Amostragem por Convêniencia, a Amostragem por Cotas, a Amostragem Intencional e a Amostragem Voluntária, ilustradas pela figura a seguir:

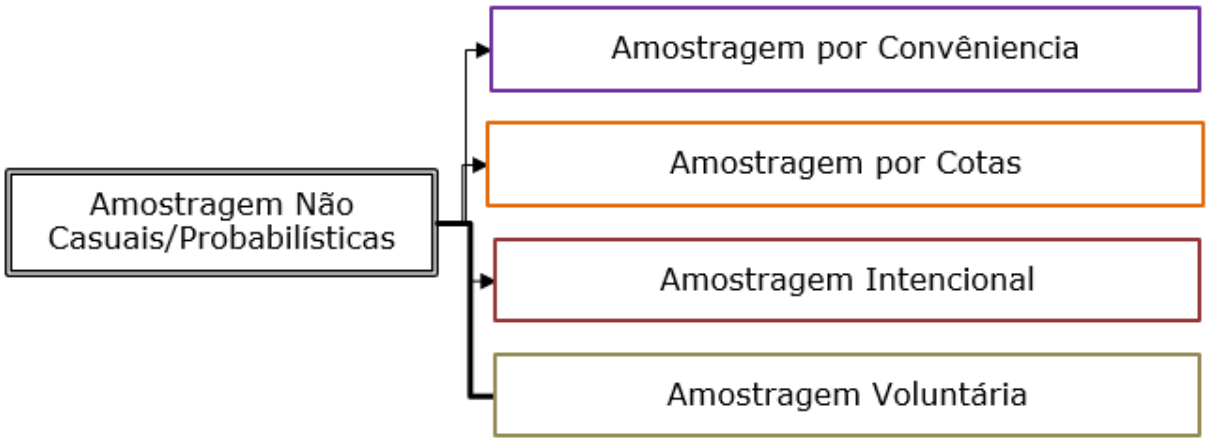


Figura 6.8: Tipos de Amostragem não casuais/não probabilísticas

Amostragem por Conveniência

Consiste na seleção dos elementos aos quais se tem acesso no momento.

Exemplo:

Imaginemos que só temos dados dos clientes do RJ para fazer uma pesquisa. Por conveniência, utilizaremos só esses dados, desconsiderando os outros municípios.

Amostragem por Cotas

A seleção é feita de acordo com determinada característica da população.

Exemplo:

A priori, é sabido que 50% das mulheres jogam vôlei em um campeonato, então se forem considerados na pesquisa 1000 atletas desse campeonato, 500 deles serão mulheres jogadoras de vôlei.

Amostragem Intencional

Consiste na seleção de elementos, de acordo com informações disponíveis, de forma a satisfazer a necessidade de quem está pesquisando.

Exemplo:

Sabemos que por opinião dos especialistas, certo grupo de empresas realizam tal padrão de fraude. Vamos então investigar essas empresas, de modo a satisfazer a minha intenção/objetivo.

Amostragem Voluntária

Consiste na seleção do indivíduo pela própria voluntariedade dele de participar da pesquisa.

Exemplo:

Foi descoberta determinada vacina para uma doença. Indivíduos portadores dela se voluntariam a participar da pesquisa, como cobaias.

6.4 Reamostragem

Vimos que, uma vez que temos uma amostra dos dados, esta pode ser usada para estimar parâmetros de toda a população. Entretanto, se temos apenas uma amostra, teremos apenas uma única estimativa do parâmetro populacional de interesse, sem termos ideia da incerteza ou da variabilidade desta estimativa. Para resolver esta questão, podemos realizar a **reamostragem**, que consiste em estimar o parâmetro populacional várias vezes a partir de uma mesma amostra de dados. Como resultado, teremos uma estimativa mais precisa e uma quantificação da incerteza da estimativa.

Na reamostragem, cada nova subamostra da amostra de dados original é usada para estimar o parâmetro populacional e podemos,

a partir destas estimativas, quantificar o valor esperado e a variância, fornecendo medidas de incerteza da estimativa. Um ponto negativo desta técnica é que, como o processo deve ser repetido várias vezes, poderá ocorrer o compartilhamento das mesmas observações entre as subamostras.

Dois métodos de reamostragem muito usados em Ciência de Dados, especialmente em Machine Learning, são:

- **Bootstrap:** muito usado quando temos uma amostra pequena dos dados. As subamostras são retiradas do conjunto de dados (amostra) com substituição (permitindo o mesmo elemento apareça mais de uma vez na subamostra). A subamostra então pode ser usada como conjunto de treino e as observações não incluídas na subamostra (conhecidas como observações *out of bag*) podem ser usadas como conjunto de teste.
- **Validação cruzada k-fold:** a amostra com *n* elementos é dividida aleatoriamente em K subconjuntos disjuntos (também chamados *folds*) com aproximadamente o mesmo número de elementos (n/K). Cada um dos K subconjuntos é usado como conjunto de teste e os restantes são reunidos em um conjunto de treino. O processo é repetido K vezes, sendo gerados e avaliados K modelos de conhecimento. Cada observação é então usada 1 vez para teste e K-1 vezes para treino. Os valores mais comuns para K são 3, 5 e 10.

Vejamos exemplos dessas técnicas em Python usando a biblioteca *scikit-learn*:

```
# Bootstrap - Os resultados podem variar devido à aleatoriedade da
amostragem!

# Imports de bibliotecas
from sklearn.utils import resample

# Amostra
amostra = [1, 2, 3, 4, 5, 6, 7]
```

```
# Vamos gerar 5 subamostras de tamanho 4
for i in range (1,6):
    # Reamostragem com boostrap
    subamostra = resample(amostra, replace=True, n_samples=4)
    print('Subamostra %d com Bootstrap: %s' % (i, subamostra))

    # Observações out of bag
    oob = [x for x in amostra if x not in subamostra]
    print('Observações OOB para %d: %s' % (i, oob))
```

A saída resultante será:

```
Subamostra 1 com Bootstrap: [7, 7, 5, 4]
Observações OOB para 1: [1, 2, 3, 6]
Subamostra 2 com Bootstrap: [6, 6, 6, 1]
Observações OOB para 2: [2, 3, 4, 5, 7]
Subamostra 3 com Bootstrap: [1, 4, 3, 7]
Observações OOB para 3: [2, 5, 6]
Subamostra 4 com Bootstrap: [1, 2, 7, 3]
Observações OOB para 4: [4, 5, 6]
Subamostra 5 com Bootstrap: [3, 7, 2, 2]
Observações OOB para 5: [1, 4, 5, 6]
```

```
# Validação cruzada 3-fold

# Imports de bibliotecas
from numpy import array
from sklearn.model_selection import KFold

# Amostra
amostra = array([1, 2, 3, 4, 5, 6, 7])

# Preparação da validação cruzada
kfold = KFold(n_splits=3, shuffle=True, random_state=7)

# Folds geradas
for train, test in kfold.split(amostra):
    print('Treino: %s, Teste: %s' % (amostra[train], amostra[test]))
```

A saída resultante será:

```
Treino: [2 4 5 7], Teste: [1 3 6]
Treino: [1 2 3 5 6], Teste: [4 7]
Treino: [1 3 4 6 7], Teste: [2 5]
```

> Observação: a biblioteca *scikit-learn* fornece diversas implementações de validação cruzada. Sugerimos consultar a documentação para mais detalhes, em: http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold. html

É importante ressaltar que a **qualidade** dos dados geralmente é mais importante do que a **quantidade** de dados quando fazemos uma estimativa ou uma modelagem com base em uma amostra. Em Ciência de Dados, a qualidade dos dados inclui integridade, consistência, limpeza e precisão das observações individuais. Detalharemos o processo de **estimação** no próximo capítulo.

6.5 Intervalos de confiança

Intervalos de confiança (IC) são uma técnica estatística usada para estimar um intervalo no qual um parâmetro de uma população (por exemplo, uma média, uma proporção, uma variância) provavelmente está contido, com um determinado nível de confiança. O IC é uma medida da incerteza em torno de uma estimativa de parâmetro. Por exemplo, se calcularmos a média de uma amostra de dados e construirmos um intervalo de confiança de 95% para essa média, o intervalo resultante dirá que temos 95% de confiança de que a verdadeira média populacional está dentro desse intervalo. O nível de confiança de 95% é uma escolha comum em

análises estatísticas e indica um alto grau de confiança na estimativa. No entanto, ainda existe uma probabilidade de 5% de que o intervalo não contenha o valor verdadeiro do parâmetro, devido à natureza probabilística das amostras de dados.

Em muitos casos, calculamos uma estimativa pontual para um parâmetro de interesse com base em uma amostra de dados. Por exemplo, podemos calcular a média de uma amostra como uma estimativa da média populacional ou, no caso de modelos de Machine Learning, podemos calcular a média das acurácias de treino obtidas em cada partição de teste quando utilizamos validação cruzada. No entanto, uma única estimativa pontual não leva em consideração a variabilidade natural dos dados. Os ICs são usados para incorporar essa incerteza estatística à estimativa, fornecendo um intervalo dentro do qual o verdadeiro valor do parâmetro provavelmente se encontra.

Exemplo: Como calcular um intervalo de confiança de 95% para a média de uma população

Suponha que você tenha uma amostra de dados e deseje calcular o intervalo de confiança de 95% para a média populacional.

- Passo 1. Colete uma amostra de dados relevante para o parâmetro de interesse (por exemplo, a média).
- Passo 2. Calcule a Média ($\bar{x}$) e o Desvio Padrão Amostral (s).
- Passo 3. Determine o Tamanho da Amostra (n).
- Passo 4. Determine o Nível de Confiança ($1 - \alpha$) desejado. Para um intervalo de confiança de 95%, α é igual a 0,05, pois a confiança é de 95%, deixando uma probabilidade de 5% para as caudas da distribuição (de acordo com a tabela de distribuição normal padrão (Z)).
- Passo 5. Calcule o Erro Padrão (SE - Standard Error) usando a fórmula: $SE = s/\sqrt{n}$, onde s é o desvio padrão amostral e n é o tamanho da amostra.
- Passo 6. Determine o valor crítico correspondente ao nível de confiança escolhido e ao tamanho da amostra. Para um intervalo de confiança de 95%, o valor crítico geralmente é aproximadamente 1,96 para uma amostra grande.
- Passo 7. Use o valor crítico e o erro padrão para calcular o intervalo de confiança: $IC = \bar{x} +- (Valor\ crítico$ SE$)$, *onde* $\bar{x}$ *é a média amostral,* Valor crítico *é o valor crítico correspondente ao nível de confiança e tamanho da amostra e* SE* é o erro padrão.

Por exemplo, se tivermos uma amostra com uma média de 50, um desvio padrão amostral de 10, um tamanho de amostra de 100 e desejarmos um intervalo de confiança de 95%, o valor crítico pode ser aproximadamente 1,96. O cálculo seria: $IC = 50 +- (1,96$ 10/$\sqrt{100}$ = 50 +- 1,96*

Assim, o intervalo de confiança de 95% corresponderia a aproximadamente (48,04, 51,96) para a média populacional.

Os intervalos de confiança são utilizados em Machine Learning como uma ferramenta para avaliar a incerteza associada às estimativas de desempenho de modelos e resultados de experimentos. Exemplos de utilização incluem:

- **Avaliação de desempenho de modelos:** ao treinar e avaliar modelos de Machine Learning, é comum calcular métricas de desempenho, como precisão, recall, F1-score, entre outras. Em vez de relatar uma única pontuação para essas métricas, os intervalos de confiança podem ser calculados para indicar a faixa de valores dentro da qual a verdadeira métrica de desempenho pode estar com um determinado nível de confiança. Isso fornece uma compreensão mais robusta do desempenho do modelo.
- **Comparação de modelos:** ao comparar diferentes modelos de Machine Learning, os intervalos de confiança podem ajudar a determinar se as diferenças observadas nas métricas de desempenho são estatisticamente significativas. Isso é importante para decidir se um modelo é realmente melhor do que outro ou se as diferenças são devidas ao acaso.
- **Avaliação de experimentos:** em experimentos de Machine Learning, como ajuste de hiperparâmetros ou seleção de modelos, os intervalos de confiança podem ser usados para quantificar a incerteza nas conclusões do experimento. Isso ajuda os praticantes a tomarem decisões informadas com base nos resultados.
- **Visualização de resultados:** os intervalos de confiança também podem ser usados para criar gráficos de barras de erro ou gráficos de intervalo que destacam a variabilidade nos resultados. Isso é útil para comunicar a incerteza aos stakeholders ou em apresentações de dados.

- **Avaliação de generalização:** ao usar validação cruzada ou divisão de dados em treinamento/teste, os intervalos de confiança podem ser calculados para estimar a variabilidade no desempenho do modelo em diferentes conjuntos de dados. Isso ajuda a avaliar a capacidade de generalização do modelo.

Vejamos um exemplo prático de como utilizar intervalos de confiança usando validação cruzada em Python, utilizando a biblioteca *scikit-learn*:

```
import numpy as np
from sklearn.model_selection import cross_val_score
from scipy import stats

# Temos um conjunto de dados e um modelo de classificação
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

# Carregando o conjunto de dados
iris = load_iris()
X = iris.data
y = iris.target

# Criando um modelo
clf = DecisionTreeClassifier()

# Realizando a validação cruzada com 5 folds
scores = cross_val_score(clf, X, y, cv=5, scoring = 'accuracy')

# Calculando a média das acurácias
mean_score = np.mean(scores)

# Calculando o desvio padrão das acurácias
std_deviation = np.std(scores)

# Calculando o intervalo de confiança de 95% para as acurácias
confidence_interval = stats.t.interval(0.95, len(scores) - 1,
loc=mean_score, scale=std_deviation / np.sqrt(len(scores)))
```

```
print("Média das Acurácias:", mean_score)
print("Desvio Padrão das Acurácias:", std_deviation)
print("Intervalo de Confiança (95%):", confidence_interval)
```

A saída resultante será:

```
Média das Acurácias: 0.9600000000000002
Desvio Padrão das Acurácias: 0.03265986323710903
Intervalo de Confiança (95%): (0.9194474236322232, 1.0005525763677772)
```

Neste exemplo, usamos o `cross_val_score` do *scikit-learn* para realizar validação cruzada em um modelo de classificação de árvore de decisão (algoritmo de Machine Learning que será detalhado no capítulo 7 deste livro). Em seguida, calculamos a média e o desvio padrão das acurácias de cada partição de teste da validação cruzada. Finalmente, usamos a função `t.interval` do módulo `scipy.stats` para calcular o intervalo de confiança de 95% para as pontuações. Este exemplo demonstra como calcular um intervalo de confiança para o desempenho de um modelo de Machine Learning usando validação cruzada. Os intervalos de confiança ajudam a quantificar a incerteza associada ao desempenho do modelo e fornecem uma estimativa da faixa em que a métrica de avaliação verdadeira provavelmente se encontra.

Neste capítulo, começamos a falar sobre Inferência Estatística, abordando as técnicas de amostragem. Vimos que as Amostragens Casuais/Probabilísticas podem ser do tipo Aleatória Simples, Sistemática, Estratificada e por Conglomerado ou Cluster. Já as Amostragens Não casuais/Não probabilísticas podem ser dos tipos por Conveniência, por Cotas, Intencional e Voluntária. Também falamos de conceitos de Reamostragem, e como esta técnica é utilizada em Machine Learning. Finalizamos falando de Intervalos de Confiança. No próximo capítulo, continuaremos estudando o tema de Inferência Estatística e falaremos sobre os principais algoritmos de Machine Learning usados para Modelagem Estatística.

CAPÍTULO 7
Inferência Estatística: Modelagem Estatística - Introdução aos algoritmos de Machine Learning e comparação de modelos

Muitas vezes queremos utilizar nossos dados para não só para fazer análises descritivas (entender o que aconteceu) ou diagnósticas (entender as razões pelas quais algo aconteceu), mas também para fazer predições, ou seja, estimar o que acontecerá. Para as análises descritivas e diagnósticas, geralmente utilizamos técnicas de Business Intelligence (BI), enquanto, para a análise preditiva, podemos utilizar técnicas de Machine Learning (ML). Este capítulo apresentará alguns dos principais algoritmos de Machine Learning, bem como um exemplo prático de comparação de modelos.

Observação: se você conhece o livro *Introdução a Data Science: Algoritmos de Machine Learning e métodos de análise* (que também é da Casa do Código e escrito por uma das autoras deste livro), pode reconhecer algumas das explicações deste capítulo, que foram adaptadas para Python aqui.

7.1 Problemas de Machine Learning

No contexto de Ciência de Dados, muitas vezes trabalhamos com modelos de Machine Learning, a fim de construirmos modelos estatísticos capazes de "aprenderem" com os nossos dados e fazerem predições. Este aprendizado pode ser categorizado em quatro tipos: **supervisionado**, quando o modelo de conhecimento é construído a partir dos dados apresentados na forma de pares ordenados (entrada, saída desejada); **não supervisionado**, quando não existe a informação da saída desejada e o processo de

aprendizado busca identificar regularidades entre os dados a fim de agrupá-los em função das similaridades que apresentam entre si; **semi-supervisionado**, que combina o aprendizado supervisionado e o não supervisionado e o aprendizado **por reforço**, quando a máquina é capaz de "perceber" o estado do ambiente, executar ações de acordo com este estado, receber "recompensas" de acordo com as ações executadas e trocar de estado, se apropriado. Neste capítulo, vamos focar no tipo mais utilizado de aprendizado de máquina: o aprendizado supervisionado, que trata dos problemas de **Classificação** e **Regressão**.

Quando temos uma pergunta como: "Devo **conceder ou não** crédito para um cliente?", temos um problema de Classificação, pois a variável a ser predita é categórica, representada pelas classes "sim" e "não". Já se a pergunta fosse "Devo conceder **qual** valor de crédito para um cliente?", teríamos um problema de Regressão, pois desejamos fazer a predição de um valor numérico (contínuo ou discreto). Para ambos os casos, podemos aprender com dados passados de clientes que fizeram empréstimos para decidir o que fazer quando chegar a solicitação de um novo cliente.

Assim, dentro do ciclo de vida de projetos de Ciência de Dados, a etapa de **Modelagem e Inferência** consiste (de forma simplificada) em escolher o modelo mais adequado para resolver o problema em questão. Resumidamente, esta etapa consiste de 3 tarefas:

1. Elencar os modelos possíveis e passíveis para cada tipo de problema;
2. Estimar os parâmetros que compõem os modelos, baseando-se nas instâncias e variáveis pré-processadas;
3. Avaliar os resultados de cada modelo, usando métricas e um processo justo de comparação

A figura a seguir ilustra o esquema básico de um projeto de Ciência de Dados que representa o seu ciclo de vida. Este esquema foi apresentado originalmente em *Introdução a Data Science:*

Algoritmos de Machine Learning e métodos de análise (Escovedo & Koshiyama, 2020), livro que traz mais detalhes sobre Ciência de Dados e especialmente algoritmos de Machine Learning.

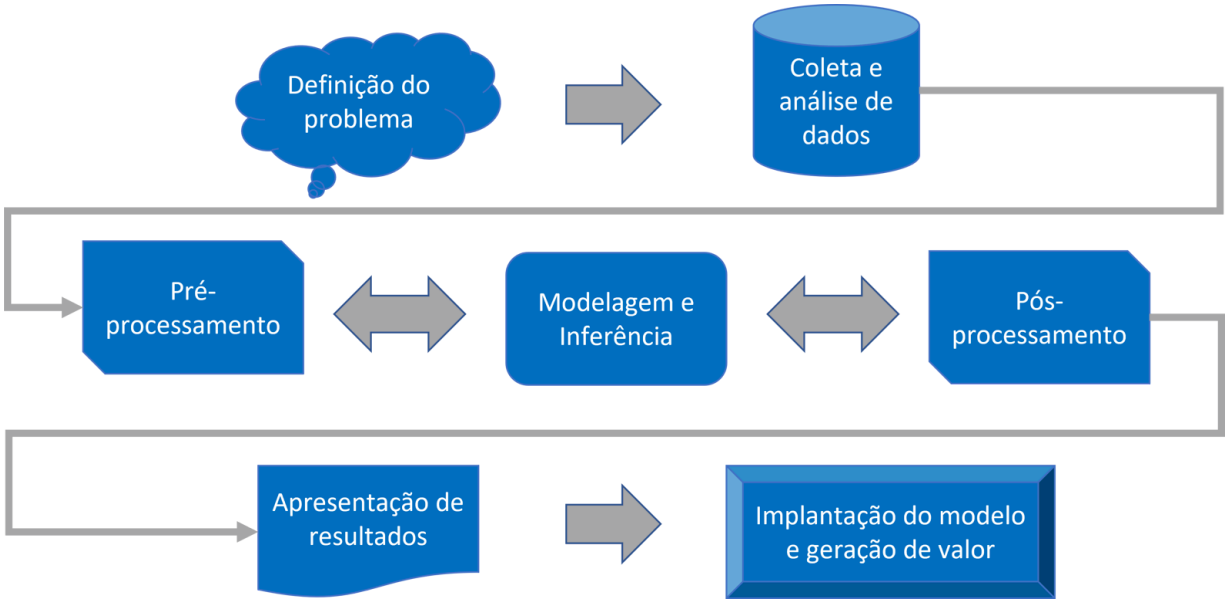


Figura 7.1: Esquema básico de um projeto de Ciência de Dados

Em problemas de aprendizado supervisionado, na etapa de modelagem, apresentamos para o algoritmo um número suficiente de exemplos (também chamados de registros ou instâncias) de entradas e saídas desejadas (já rotuladas previamente). O objetivo do algoritmo é aprender uma regra geral que mapeie as entradas nas saídas corretamente. Os dados de entrada podem ser divididos em dois grupos: *X*, com os atributos a serem utilizados na determinação da classe de saída (também chamados de atributos *independentes*, *previsores* ou de *predição*) e *y*, com o atributo para o qual se deseja fazer a predição do valor de saída (também chamado de *atributo-alvo*, *atributo dependente* ou *target*). Em problemas de Classificação, o atributo-alvo é sempre categórico (classe) e em problemas de Regressão, numérico (discreto ou contínuo), e para ambos os problemas, dado um novo padrão, espera-se que o modelo estime o valor ou classe mais esperada para a variável resposta.

Existem diversas medidas para estimar o desempenho de um modelo de aprendizado supervisionado. Para problemas de Classificação, a mais utilizada é a **acurácia**, ou taxa de acerto do classificador, dada por:

$$Acc(h) = 1 - Err(h)$$

A acurácia é uma função da taxa de erro (ou taxa de classificação incorreta), dada por:

$$Err(h) = \frac{1}{n}\sum_{i=1}^{n} \|y_i \neq h(i)\|$$

Onde:

- O operador //*E*// retorna 1 se a expressão *E* for verdadeira e 0 em caso contrário;
- *n* é o número de exemplos (registros da base de dados);
- y_i é a classe real associada ao i-ésimo exemplo;
- *h(i)* é a classe indicada pelo classificador para o i-ésimo exemplo.

Já para problemas de Regressão, em vez de se estimar a acurácia, estima-se a **distância** ou o **erro** entre a saída do estimador (modelo) e a saída desejada. A saída de um estimador é um valor numérico contínuo que deve ser o mais próximo possível do valor desejado, e a diferença entre esses valores fornece uma medida de erro de estimação do algoritmo.

Vale a pena mencionar o teorema "não existe almoço grátis": não existe um algoritmo de aprendizado que seja superior a todos os demais quando considerados todos os problemas possíveis. A cada

problema, os algoritmos disponíveis devem ser experimentados a fim de identificar aqueles que obtêm melhor desempenho.

7.2 Algoritmos de Machine Learning

A seguir, serão apresentados alguns algoritmos de Machine Learning que podem ser usados para problemas de Classificação e Regressão.

KNN (Classificação e Regressão)

O algoritmo KNN (*k-Nearest Neighbours* ou, em português, k-Vizinhos Mais Próximos) é simples de entender e funciona muito bem na prática tanto para problemas de Classificação quanto para problemas de Regressão. Este é um algoritmo não paramétrico, ele não assume premissas sobre a distribuição dos dados. Sua ideia principal é considerar que os exemplos vizinhos são similares ao exemplo cuja informação se deseja inferir. Assim, o KNN utiliza uma métrica de distância para encontrar as *k* instâncias mais semelhantes nos dados de treinamento para uma nova instância e considera o resultado dos vizinhos como a previsão (a classe mais comum, em problemas de classificação, ou a média da variável *target*, em problemas de regressão).

A figura a seguir ilustra esta ideia para problemas de classificação: o KNN considera que os registros do conjunto de dados correspondem a pontos no $\mathbf{R_n}$, em que cada atributo corresponde a uma dimensão deste espaço.

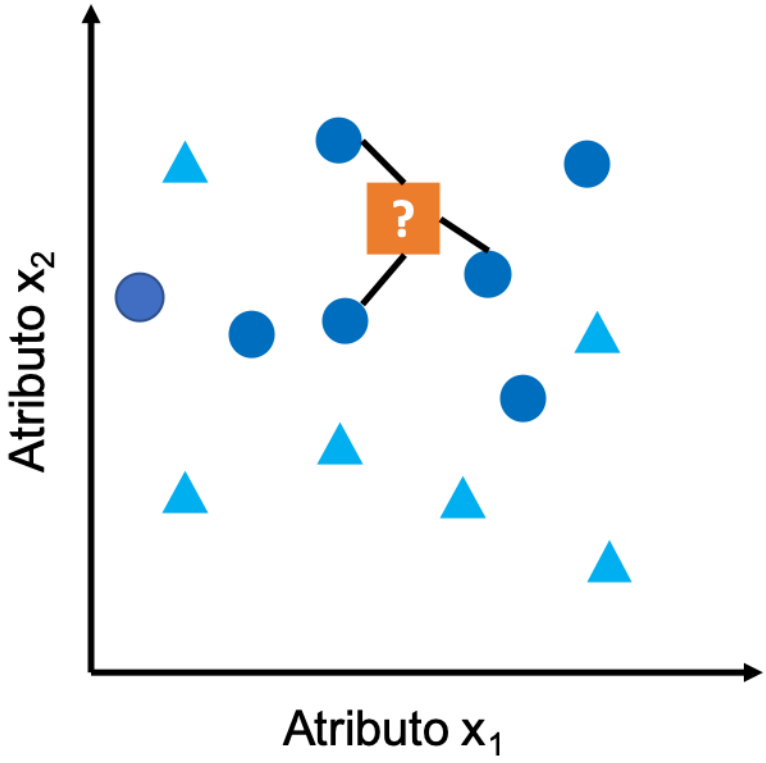


Figura 7.4: Exemplo do funcionamento do KNN

No KNN, inicialmente, o conjunto de dados (rotulado) é armazenado. Quando um novo registro deve ser classificado, ele é comparado a todos os registros do conjunto de treinamento para identificar os **k** (parâmetro de entrada) vizinhos mais próximos (mais semelhantes) de acordo com alguma métrica de distância (por exemplo, distância Euclidiana). A classe do novo registro é determinada por inspeção das classes desses vizinhos mais próximos, de acordo com a métrica escolhida. Na maioria das implementações do KNN, os atributos são normalizados, para que tenham a mesma contribuição na predição da classe ou do valor.

As etapas a seguir resumem o algoritmo KNN:

1. Definição da métrica de distância e valor de **k**.
2. Cálculo da distância do novo registro a cada um dos registros existentes no conjunto de referência.

3. Identificação dos **k** registros do conjunto de referência que apresentaram menor distância em relação ao novo registro (mais similares).
4. Apuração da classe mais frequente entre os **k** registros identificados no passo anterior (usando votação majoritária), para problemas de Classificação, ou da média aritmética dos *k*-vizinhos mais próximos, para problemas de Regressão.

Apesar de ser um algoritmo muito utilizado, o KNN tem algumas limitações: a performance de predição pode ser lenta em datasets grandes; é sensível a características irrelevantes, uma vez que todas as características contribuem para o cálculo da distância e consequentemente, para a predição; e é necessário testar diferentes valores de **k** e a métrica de distância a utilizar.

Árvore de Decisão (Classificação e Regressão)

A Árvore de Decisão é um dos modelos preditivos mais simples de ser interpretado, e é inspirada na forma como humanos tomam decisão. Uma de suas principais vantagens é apresentar a informação visualmente, de uma forma fácil de entender pelo ser humano. As árvores podem ser usadas para problemas de Classificação (chamadas de Árvores de Classificação) ou Regressão (chamadas Árvores de Regressão). Basicamente, uma árvore de decisão usa amostras das características dos dados para criar regras de decisão no formato de árvore, isto é, reduz os dados em um conjunto de regras que podem ser usadas para uma decisão.

As árvores de decisão aliam acurácia e interpretabilidade. Elas possibilitam a seleção automática de variáveis para compor suas estruturas: cada nó interno representa uma decisão sobre um atributo que determina como os dados estão particionados pelos seus nós filhos. Para fazer a predição de um novo exemplo, basta testar os valores dos atributos na árvore e percorrê-la até se atingir um nó folha (resultado predito). A figura a seguir ilustra uma Árvore de Classificação.

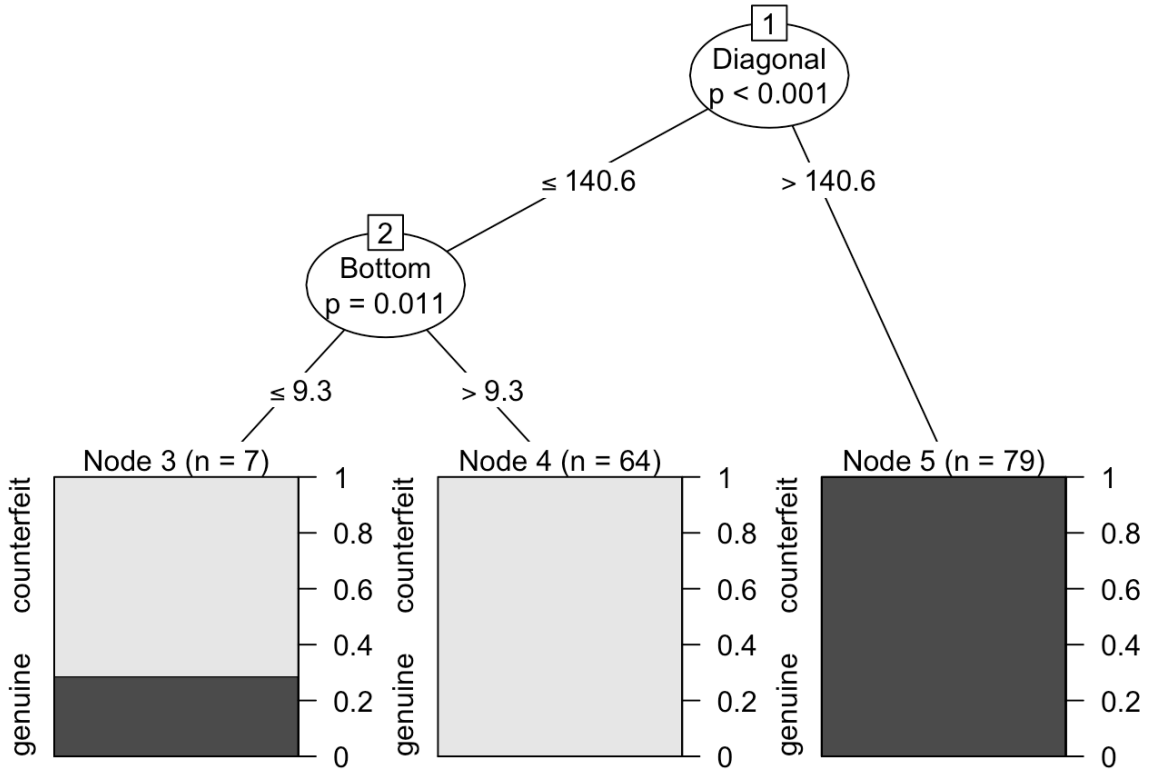


Figura 7.5: Exemplo de Árvore de Classificação

As Árvores de Regressão são muito similares às Árvores de Classificação, com a diferença de que, nas Árvores de Regressão, a predição (ou valor estimado) é a média dos valores dos exemplos de cada folha, enquanto nas Árvores de Classificação a predição é a classe mais frequente nos exemplos de cada folha. Ambas são construídas de forma similar: a partir do nó raiz, os dados são particionados usando uma estratégia de divisão e conquista de acordo com a característica que resultará no resultado mais homogêneo após a separação ser realizada. Enquanto nas Árvores de Classificação a homogeneidade é medida pela entropia (ou medidas similares), nas Árvores de Regressão, a homogeneidade é medida por estatísticas como variância, desvio padrão ou desvio absoluto da média.

Há diferentes algoritmos para a elaboração de uma Árvore de Decisão. Alguns exemplos são: ID3, CTree, C4.5, C5.0 e CART. Todos os algoritmos são bem parecidos: em geral, a construção da árvore

é realizada de acordo com alguma abordagem recursiva de particionamento do conjunto de dados, e a principal distinção está nos processos de seleção de variáveis, critério de particionamento e critério de parada para o crescimento da árvore.

Embora sejam intuitivas e interpretáveis, as árvores de decisão por si só não costumam ter o mesmo nível de precisão preditiva que alguns outros algoritmos populares de aprendizado de máquina. Elas tendem a criar limites de decisão excessivamente complicados, resultando em maior variação do modelo, o que leva ao *overfitting*. Para evitar este problema na prática, podemos usar algoritmos de poda (*pruning*) para reduzir a profundidade e a complexidade da árvore, removendo nós maiores que uma certa profundidade, para que a árvore de decisão não se ajuste em excesso aos dados de treinamento.

Árvore C4.5

Dentre os algoritmos de Árvores de Classificação existentes, um dos mais conhecidos é o C4.5, que utiliza conceitos e medidas de Teoria da Informação, uma disciplina que estuda a quantificação, armazenamento e comunicação da informação. O funcionamento do C4.5 pode ser resumido conforme a seguir: inicialmente, a raiz da árvore contém todo o conjunto de dados com exemplos misturados de várias classes. Um predicado (ponto de separação) é escolhido como sendo a condição que melhor separa ou discrimina as classes. Um predicado é um dos atributos previsores do problema e induz uma divisão do conjunto de dados em dois ou mais conjuntos disjuntos, cada um deles associado a um nó filho. Cada novo nó abrange um subconjunto do conjunto de dados original que é recursivamente separado até que o subconjunto associado a cada nó folha consista inteira ou predominantemente de registros de uma mesma classe.

As etapas a seguir resumem o algoritmo C4.5 para problemas de Classificação:

1. Calcular a entropia do conjunto **T** completo.
2. Para cada atributo:
    - Calcular o ganho de informação.
3. Selecionar o atributo com maior ganho de informação para o nó raiz da árvore.
4. Subdividir o conjunto **T**.
5. Repetir o procedimento para cada nó gerado.

Naïve Bayes (Classificação)

O Naïve Bayes, ou Bayes Ingênuo, é um classificador genérico e de aprendizado dinâmico. É um dos métodos mais utilizados para Classificação - especialmente em aplicações de *text mining*, previsões em tempo real e/ou em sistemas embarcados - pois é rápido computacionalmente e só necessita de um pequeno número de dados de treinamento. Ele é especialmente adequado quando o problema tem um grande número de atributos (características), e determina a probabilidade de um exemplo pertencer a uma determinada classe.

Este método é chamado de ingênuo (*naïve*, em inglês) porque desconsidera completamente a correlação entre os atributos (características), tratando cada um de forma independente. Além disso, o nome do método contém a palavra Bayes porque é baseado no Teorema de Bayes, que determina a probabilidade de um evento com base em um conhecimento prévio (a priori) que pode estar relacionado a este evento.

Seja X uma observação (ou evidência) proveniente de um conjunto de n atributos. Seja *H* a hipótese que a observação *X* pertença a uma determinada classe *C*. Em problemas de Classificação, queremos determinar *P(H|X)*, ou seja, a probabilidade que a hipótese *H* se aplique a *X*. Em outras palavras, buscamos a probabilidade de que a observação *X* pertença à classe *C*, dado que conhecemos a descrição de *X*. Assim:

- *P(H|X)* é denominada a probabilidade *a posteriori* de *H* condicionada a *X*. Por exemplo, suponha que tenhamos os dados de idade e renda mensal de clientes, *X* é um cliente de 30 anos com renda mensal R$ 5.000,00 e *H* é a hipótese de este cliente comprar um celular. *P(H|X)* é então a probabilidade de este cliente *X* comprar um celular dado que conhecemos sua idade e renda mensal.
- *P(H)*, por sua vez, é a probabilidade a priori de *H*. É a probabilidade de qualquer cliente, independente de idade, renda mensal ou qualquer outra informação, comprar um celular. Esta probabilidade é baseada em mais informação do que *P(H)*, que é independente de *X*.
- De forma análoga, *P(X|H)* é a probabilidade a posteriori de *X* condicionada a *H*. Por exemplo, a probabilidade de um cliente *X* ter 30 anos e ganhar R$ 5.000,00 dado que este cliente comprou um celular.
- *P(X)*, finalmente, é a probabilidade a priori de *X*, a probabilidade que um cliente do nosso conjunto de dados tenha 30 anos e ganhe R$ 5.000,00.

P(H), *P(X|H)* e *P(X)* podem ser estimadas a partir do próprio conjunto de dados. O Teorema de Bayes é útil para calcular a probabilidade a posteriori *P(H|X)* a partir de *P(H)*, *P(X|H)* e *P(X)*:

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)}$$

Finalmente, apresentamos a definição formal do algoritmo Naïve Bayes. Seja $X(A_1, A_2, \ldots, A_n, C)$ um conjunto de dados. Considere que $c_1, c_2, \ldots, c_k$ são as classes do problema (valores possíveis do atributo alvo *C*) e que *R* é um registro que deve ser classificado. Sejam ainda $a_1, a_2, \ldots, a_n$ os valores que *R* assume para os atributos previsores $A_1, A_2, \ldots, A_n$, respectivamente.

O algoritmo consiste em dois passos:

1. Calcular as probabilidades condicionais $P(C=c_i|R), i = 1, 2, \ldots, k$
2. Indicar como saída do algoritmo a classe *c* tal que $P(C=c|R)$ seja máxima, quando considerados todos os valores possíveis do atributo alvo *C*.

A intuição por trás do algoritmo é dar mais peso para as classes mais frequentes e, conforme já discutimos, é dito ingênuo porque considera como hipótese que os atributos são estatisticamente independentes entre si, o que em muitos casos práticos, não ocorre. Entretanto, na prática, o método mostra-se bastante efetivo, mesmo nos casos em que os previsores não são estatisticamente independentes.

SVM (Classificação e Regressão)

O SVM (*Support Vector Machine* ou Máquina de Vetor de Suporte) é um dos algoritmos mais efetivos para Classificação e que também pode ser utilizado para problemas de Regressão, apesar de menos comum. O SVM pode ser aplicado em dados lineares ou não lineares. Apesar de o treinamento dos modelos de SVM costumar ser lento, estes modelos exigem poucos ajustes, tendem a apresentar boa acurácia e conseguem modelar fronteiras de decisão complexas e não lineares. Além disso, são menos propensos a *overfitting* se comparados com outros métodos.

Essencialmente, o SVM realiza um mapeamento não linear para transformar os dados de treino originais em uma dimensão maior. Nesta nova dimensão, o algoritmo busca pelo hiperplano que separa os dados linearmente de forma ótima. Com um mapeamento apropriado para uma dimensão suficientemente alta, dados de duas classes podem ser sempre separados por um hiperplano. O SVM encontra este hiperplano usando vetores de suporte (exemplos essenciais para o treinamento) e margens, definidas pelos vetores de suporte.

Para melhor entendimento do SVM aplicado a problemas de Classificação (uso mais comum), considere um conjunto de dados de treinamento linearmente separável na forma $\{x_i, y_i\}$, em que x_i corresponde ao vetor de 2 atributos previsores e $y_i \in \{-1,1\}$, as duas classes possíveis do problema. O conjunto de dados de entrada é utilizado para construir uma função de decisão $f(x)$ tal que:

- Se $f(x) > 0$, então $y_i = 1$
- Se $f(x) < 0$, então $y_i = -1$

A figura a seguir ilustra este exemplo.

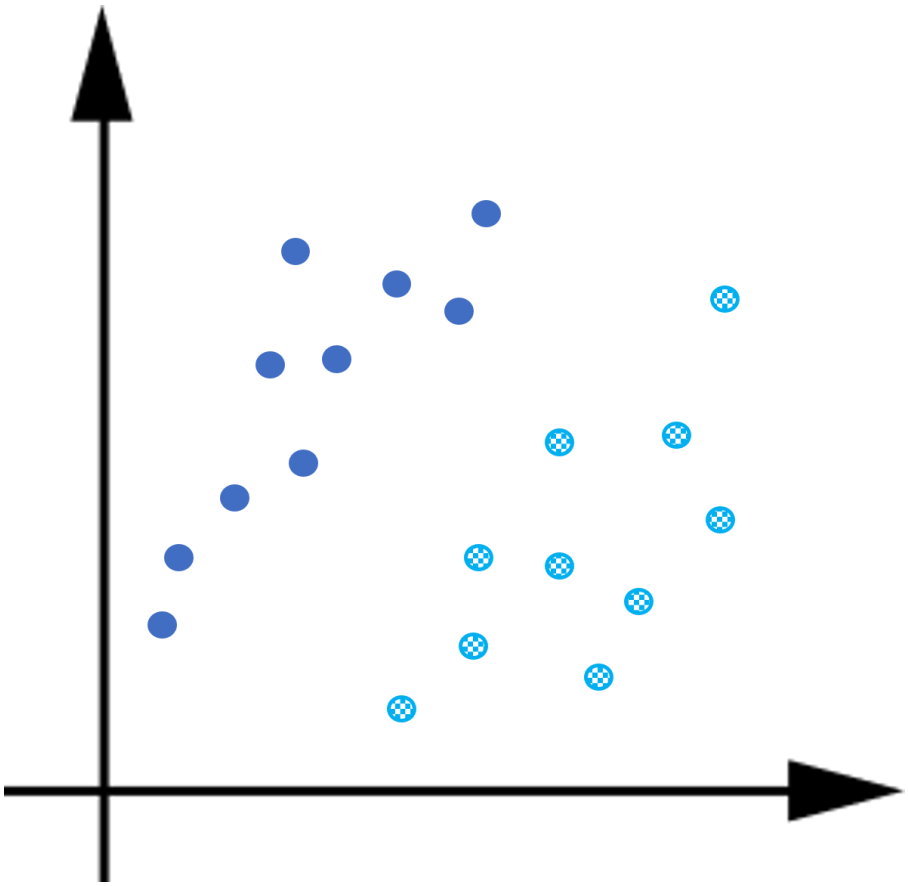


Figura 7.7: Exemplo de problema

Conforme já mencionado, o algoritmo SVM constrói classificadores lineares, que separam os conjuntos de dados por meio de um hiperplano. Considere hiperplano como a generalização do conceito

de plano para dimensões maiores que 3. Em um espaço p-dimensional, um hiperplano é um subespaço achatado de dimensão *p-1* que não precisa passar pela origem.

Quando d = 2, o hiperplano é uma reta e sua equação é dada por:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$$

onde β_1 e β_2 são parâmetros da reta que determinam a inclinação da reta e β_0 o parâmetro que determina o ponto de corte do eixo y. Neste caso, o propósito do SVM é determinar parâmetros da reta, β_0, β_1 e β_2, que permitem separar os conjuntos dos dados de treinamento em duas classes possíveis:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 < 0$$

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 > 0$$

Este exemplo é ilustrado pela figura a seguir.

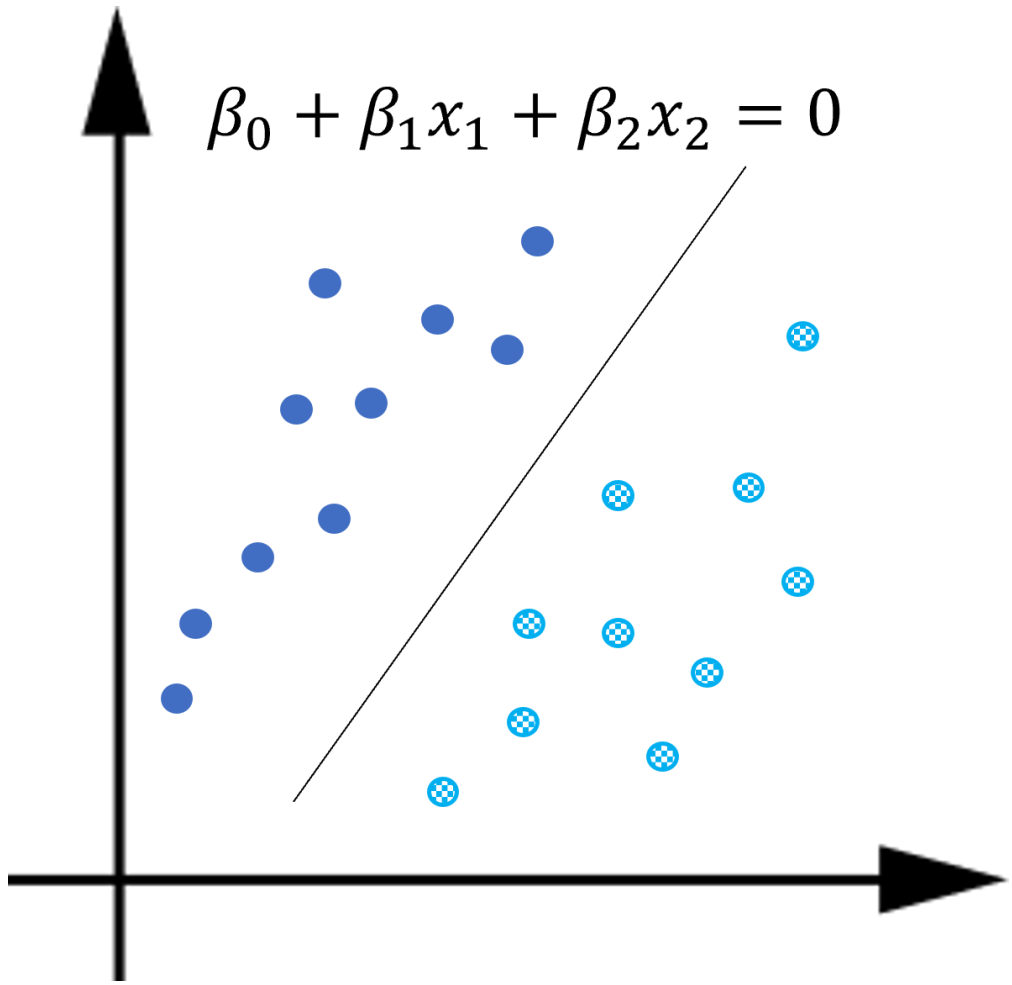


Figura 7.10: Separação dos dados em duas classes através de uma reta

Da mesma forma que um plano tem 2 dimensões e divide um conjunto tridimensional em 2 espaços de dimensão 3, um hiperplano em um espaço n-dimensional tem n-1 dimensões e divide este espaço em 2 subespaços de dimensão n. Assim, para d = p, a equação do hiperplano é:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ \ldots + \beta_n x_n = 0$$

Neste caso, o hiperplano também divide o espaço p-dimensional em 2 metades:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ \ ... + \beta_n x_n < 0$$

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ \ ... + \beta_n x_n > 0$$

Voltando ao exemplo anterior, como afirmamos previamente que o problema é linearmente separável, é possível construir um hiperplano que separe as observações de treino perfeitamente de acordo com seus rótulos de classe. Este hiperplano tem as seguintes propriedades:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 > 0, \ se \ y_i = 1$$

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 < 0, \ se \ y_i = -1$$

O hiperplano pode então ser usado para construir um classificador: o exemplo de teste recebe a classe dependendo de que lado do hiperplano estiver localizado.

É importante notar que, no exemplo anterior, infinitas retas dividem corretamente o conjunto de treinamento em duas classes, conforme ilustra a figura a seguir. O SVM deve, então, realizar um processo de escolha da reta separadora, dentre o conjunto infinito de retas possíveis.

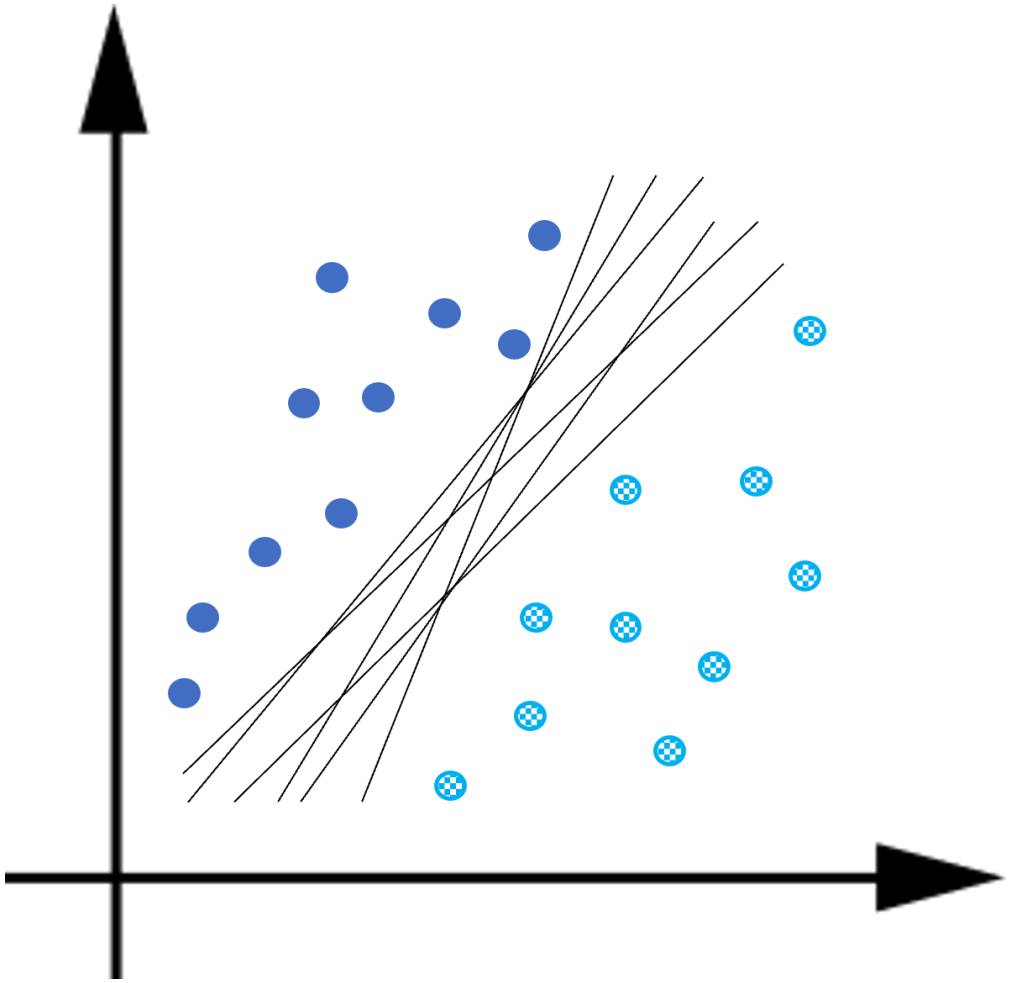


Figura 7.14: Separação dos dados em duas classes através de várias retas

Na figura a seguir, é apresentado apenas um classificador linear (ilustrado pela reta sólida) e duas retas paralelas a este classificador, pontilhadas. Cada uma das retas pontilhadas é movida a partir da posição da reta sólida, e determina quando a reta paralela intercepta o primeiro ponto do conjunto de dados. A margem é a distância construída entre estas duas retas paralelas pontilhadas.

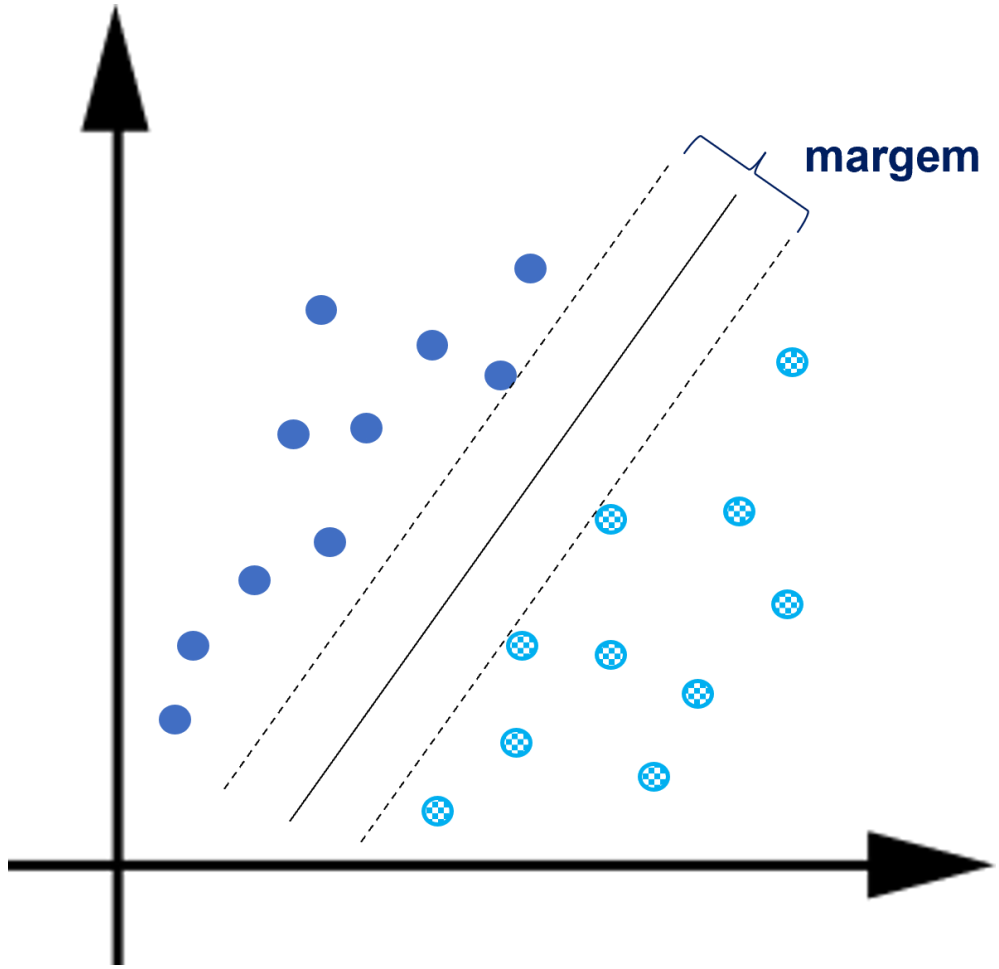


Figura 7.15: Classificador linear e margem

Assim como existem infinitas retas que separam os pontos em duas classes, há diversos tamanhos de margem possíveis dependendo da reta escolhida como classificador. A figura a seguir ilustra dos possíveis classificadores, com dois tamanhos de margem diferentes.

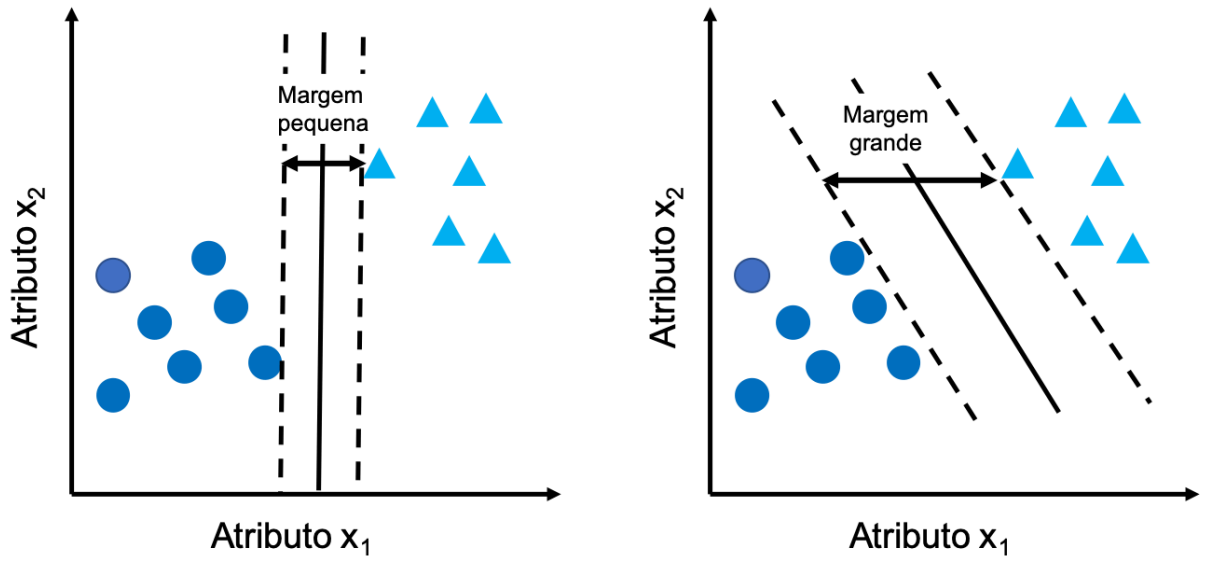


Figura 7.16: Diferentes tamanhos de margem

O classificador associado ao valor máximo de margem é denominado *classificador linear de margem máxima*. Os pontos do conjunto de dados de treinamento que são interceptados pelas linhas da margem são denominados vetores de suporte e são os pontos mais difíceis de classificar. Os vetores de suporte são ilustrados pela figura a seguir. Por construção, todos os vetores de suporte possuem a mesma distância em relação a reta do classificador linear (a metade do comprimento da margem). É importante ressaltar que, apesar de o classificador linear de margem máxima ser geralmente bom, pode haver *overfitting* se o número de dimensões for grande.

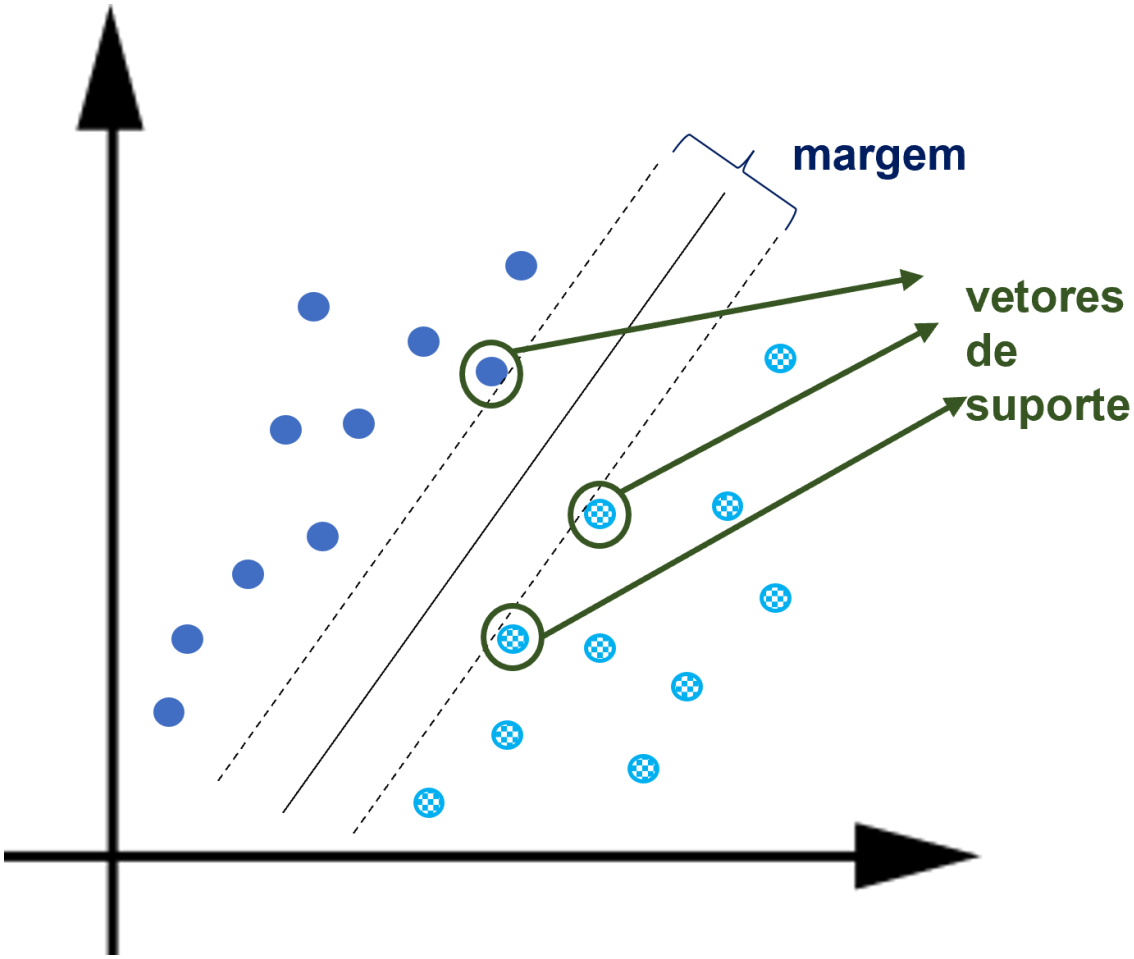


Figura 7.17: Vetores de suporte

O SVM realiza um processo de otimização, por meio do qual são determinados os parâmetros do classificador linear (β_0, β_1 e β_2). O objetivo deste processo de otimização é determinar os valores de β_0, β_1 e β_2 que produzam o valor máximo para o comprimento da margem.

A reta correspondente ao classificador linear é dita ótima porque, se ela for deslocada em alguma das duas direções das retas perpendiculares a ela, a probabilidade é maior de haver um erro de classificação. Assim, a posição do classificador linear correspondente ao comprimento de margem máximo é a mais segura possível com relação a eventuais erros de classificação, e quanto maior a distância

de *x* para o hiperplano, maior a confiança sobre a classe a que *x* pertence.

Observação: é importante ressaltar que na biblioteca *scikit-learn* o parâmetro *C* representa um parâmetro de regularização e funciona de forma contrária, adicionando uma penalidade a cada ponto classificado erradamente. Se o valor de *C* for pequeno, então, essencialmente, a penalidade para pontos classificados erradamente também será pequena, resultando em uma margem maior (mais flexível).

A solução do problema de otimização do SVM pode ser obtida usando técnicas de programação quadrática, muito conhecidas na área de Pesquisa Operacional, que consistem em otimizar uma função quadrática sujeita a restrições lineares. A solução deste problema está fora do escopo deste livro.

Uma vez obtidos os valores dos parâmetros β_0, β_1 e β_2, aplica-se uma função de decisão para classificar um novo exemplo x_i. A classe de x_i, y_i, é dada pelo sinal de *f(x)*:

$$Se\ f(x_i) > 0, então\ y_i = 1$$

$$Se\ f(x_i) < 0, então\ y_i = -1$$

Na prática, entretanto, os dados reais não costumam ser perfeitamente separáveis por um hiperplano. Além disso, o Hiperplano Margem Máxima é extremamente sensível a mudanças em uma única observação, o que sugere que pode ocorrer *overfitting* nos dados de treino. Por isso, podemos querer considerar um classificador baseado em um hiperplano que não separe perfeitamente as duas classes, com o objetivo de aumentar a robustez nas observações individuais e melhorar a classificação na maioria das observações de treino. Ou seja, pode valer a pena

classificar erroneamente algumas observações de treino a fim de melhorar a classificação nas observações restantes.

Para tal, usamos o classificador *soft-margin*, que permite que algumas observações do conjunto de treino violem a linha de separação. Neste caso, o hiperplano é escolhido para separar corretamente a maior parte das observações em duas classes, mas pode classificar incorretamente algumas observações. Um conjunto adicional de coeficientes é introduzido na otimização para permitir uma "folga" à margem. Em contrapartida, aumentam a complexidade do modelo.

No exemplo ilustrado pela figura a seguir, o classificador cometeria erros na classificação para os dois pontos destacados, que podem ser considerados ruído. Para tal, o parâmetro de custo C define a "rigidez" da margem e controla o *trade-off* entre o tamanho da margem e o erro do classificador. Quanto maior o valor de C, mais pontos podem ficar dentro da margem e maior o erro de classificação, mas menor é a chance de *overfitting*. Se $C = 0$, a margem é rígida e temos o Classificador de Margem Máxima. Na prática, este classificador não produz bons resultados.

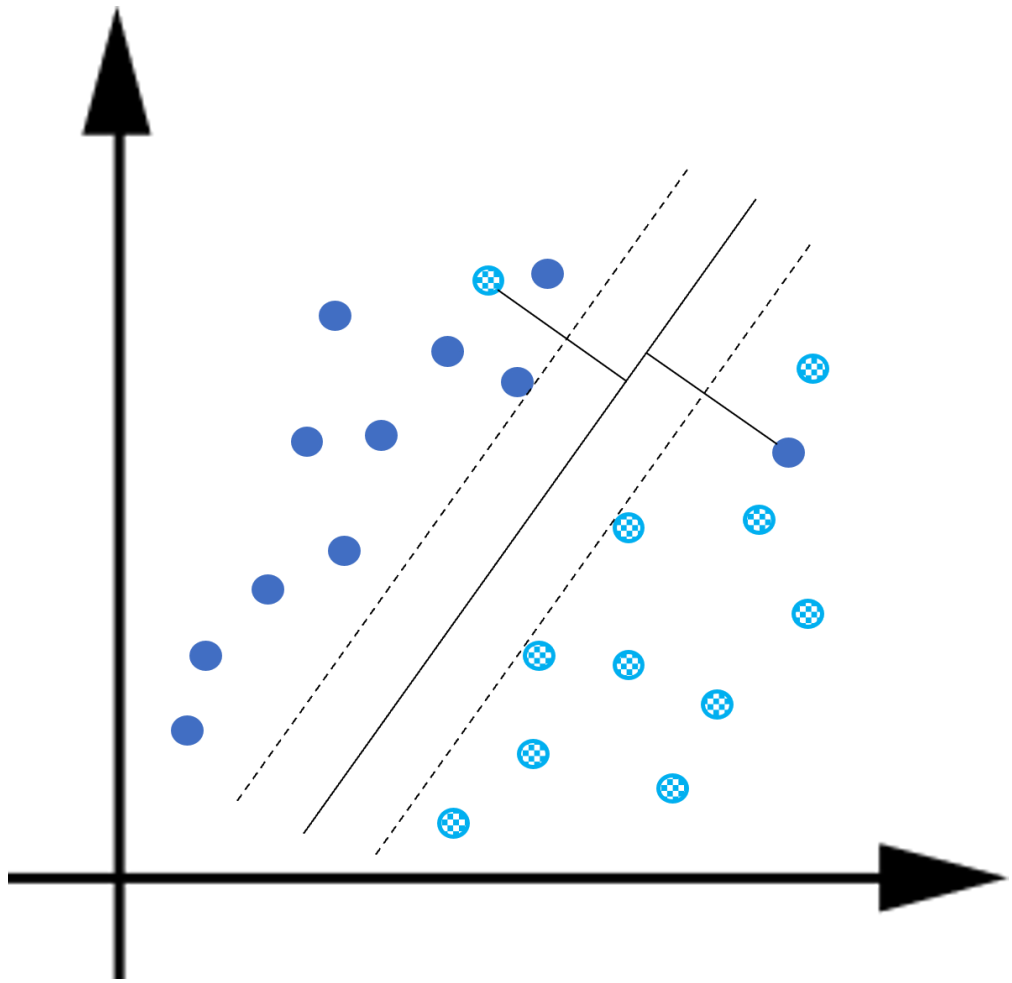


Figura 7.19: Tratamento de ruído

O número total de vetores de suporte depende da quantidade de folga permitida nas margens e da distribuição dos dados. Quanto maior a folga permitida, maior o número de vetores de suporte e mais lenta será a classificação dos dados de teste, pois a complexidade computacional do SVM está relacionada com o número de vetores de suporte.

Na prática, o SVM é implementado usando funções kernel, que, de forma simplificada, são objetos matemáticos que permitem que trabalhemos um espaço de dimensão maior. Neste caso, em vez de se utilizar as observações em si, é utilizado o seu produto interno. A previsão de um novo exemplo é feita calculando o produto escalar

entre o exemplo (x) e cada vetor de suporte (x_i). Os tipos de kernel mais utilizados são linear, polinomial e radial.

Para um conjunto de dados que não é linearmente separável, o SVM utiliza funções kernel para mapear o conjunto de dados para um espaço de dimensão maior que a original. O classificador é então ajustado neste novo espaço. O SVM é, na verdade, a combinação do classificador linear com um kernel não linear. Este processo é ilustrado pela figura a seguir.

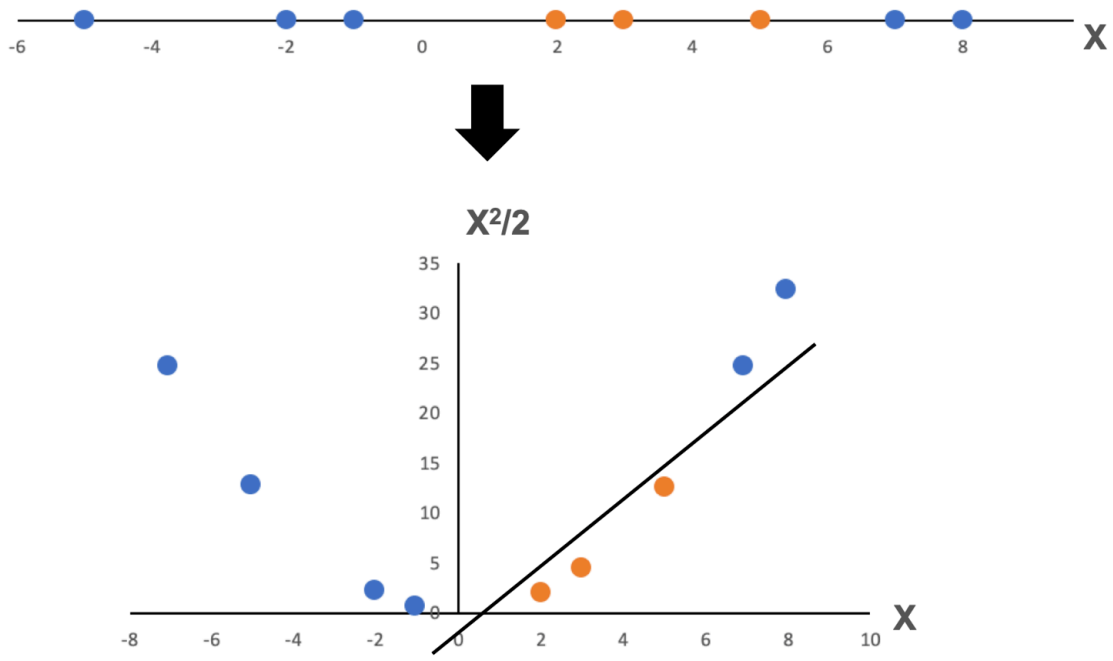


Figura 7.20: Mapeamento de um conjunto não linearmente separável em um linearmente separável

O SVM também pode ser aplicado a problemas de Classificação que envolvem múltiplas classes. Neste caso, usa-se o algoritmo para treinar um modelo de Classificação que informe se o registro é da classe c_i (região positiva) ou se é de alguma outra classe diferente de c_i (região negativa).

Desta forma, constrói-se *p-1* modelos de Classificação, onde *p* é o número de classes possíveis no problema. Para classificar um novo exemplo, pode-se submetê-lo a cada um dos modelos de classificação gerados, e a classe selecionada para o exemplo é a mais frequentemente atribuída.

Regressão Linear (Regressão)

A Regressão Linear, como o nome já diz, é um algoritmo para o problema de Regressão. Formalmente, este algoritmo modela a relação entre a variável de resposta (*y*) e as variáveis preditoras (*X*). A Regressão corresponde ao problema de estimar uma função a partir de pares entrada-saída e considera que *y* pode ser explicado por uma combinação linear de *X*. Quando temos apenas um *x* em *X*, temos uma Regressão Linear simples, cuja equação é $\mathbf{y = \beta_0 + \beta_1 x}$, sendo β_0 e β_1 os coeficientes de regressão (especificam, respectivamente, o intercepto do eixo y e a inclinação da reta). Para o caso da Regressão Linear múltipla, a equação deve ser estendida para equação de plano/hiperplano: $\mathbf{y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + + \beta_n x_n}$.

Ou seja, a solução da tarefa de regressão consiste em encontrar valores para os coeficientes de regressão de forma que a reta (ou plano/hiperplano) se ajuste aos valores assumidos pelas variáveis no conjunto de dados. Além do Método dos Mínimos Quadrados, existem diversas técnicas matemáticas para determinar os coeficientes, que estão fora do escopo deste livro.

A saída do estimador é um valor numérico contínuo que deve ser o mais próximo possível do valor desejado, e a diferença entre esses valores fornece uma medida de erro de estimação do algoritmo. Então, seja $d_j = 1, \ldots, n$ a resposta desejada para o objeto *j* e y_j a resposta predita do algoritmo, obtida a partir da entrada x_j. Então, $e_j = d_j - y_j$ é o erro observado na saída do sistema para o objeto *j*.

O processo de treinamento do estimador tem por objetivo corrigir este erro observado e, para tal, busca minimizar um critério (função objetivo) baseado em e_j, de maneira que os valores de y_j estejam próximos dos de d_j no sentido estatístico. Se a equação de regressão aproxima suficiente bem os dados de treinamento, então ela pode ser usada para estimar o valor de uma variável (y) a partir do valor da outra variável (X), assumindo uma relação linear entre as estas variáveis. Em suma, a Regressão Linear procura pelos coeficientes da reta que minimizam a distância dos objetos à reta.

Regressão Logística (Classificação)

A Regressão Logística, apesar do nome, é um algoritmo utilizado para problemas de Classificação, apesar do seu funcionamento lembrar muito o algoritmo de Regressão Linear. O algoritmo da Regressão Logística é usado para estimar valores discretos (valores binários como 0/1, sim/não, verdadeiro/falso) com base em um conjunto de variáveis independentes. Internamente, a Regressão Logística calcula a probabilidade de ocorrência de um evento, ajustando os dados a uma função *logit* (por isso, também é conhecido como regressão *logit*). Esta função pode ser descrita por:

$$\operatorname{logit}(p) = \log\left(\frac{p}{1-p}\right) = \log(p) - \log(1-p).$$

Como prevê a probabilidade, seus valores de saída estão entre 0 e 1 (como esperado). Este algoritmo utiliza a função logística, também chamada de sigmóide: uma curva em forma de S que pode mapear qualquer número em um intervalo entre 0 e 1 (mas nunca exatamente nestes limites), dada por:

$$f(x) = \frac{1}{1+e^{-x}}$$

onde *f(x)* é a saída prevista, β_0 é o intercepto e β_1 é o coeficiente para o atributo *x*, que deve ser aprendido a partir dos dados de treino.

De forma similar à Regressão Linear, a Regressão Logística usa uma equação como representação: os valores de entrada (*X*) são combinados linearmente usando pesos ou valores de coeficiente para prever um valor de saída (*y*). A diferença é que o valor de saída é modelado em valor binário (0 ou 1) em vez de um valor numérico.

A Regressão Logística modela a probabilidade da classe padrão do problema. Por exemplo, se estivermos modelando o sexo de uma pessoa dada sua altura e considerarmos a classe "masculino" como padrão, o modelo pode ser escrito como: **P(sexo = masculino | altura)**. Escrito de outra forma, estamos modelando a probabilidade de uma entrada X pertencer à classe padrão (Y = 1): P(X) = **P(Y=1 | X)**. Note que a predição da probabilidade pode ser transformada em um valor binário (0 ou 1) para fazer a classificação.

Os coeficientes da Regressão Logística podem ser estimados usando os dados de treinamento, através do método de estimação de máxima verossimilhança, um método matemático que busca valores para os coeficientes de forma a minimizar o erro nas probabilidades preditas pelo modelo para os dados, cujo detalhamento está fora do escopo deste livro. Os melhores coeficientes resultarão em um modelo que vai prever um valor muito próximo de 1 para a classe padrão e um valor muito próximo de 0 para a outra classe.

Após determinados os coeficientes, para fazer predições em novos dados com a Regressão Logística, basta calcular os coeficientes e aplicar a equação resultante.

7.3 Exemplo prático

Para ilustrar como aplicamos os algoritmos de Machine Learning na prática, vamos examinar o popular *dataset* Wine (https://archive.ics.uci.edu/ml/datasets/Wine), extraído a partir de uma análise química de vinhos cultivados na mesma região da Itália, mas derivados de três produtores diferentes. O objetivo desse dataset é identificar o produtor do vinho com base em 13 características químicas do vinho, ou seja, é um problema de Classificação. O *dataset* contém 178 instâncias (linhas), sendo 59 do produtor 1, 71 do do produtor 2 e 48 do produtor 3.

Vejamos um exemplo de código em Python usando a biblioteca *scikit-learn*. Primeiramente, vamos carregar o *dataset* e separar em bases de treino e teste através do método *holdout*. Em seguida, para a base de treino, vamos avaliar a acurácia dos modelos treinados com os algoritmos Regressão Logística, KNN, Árvore de Classificação, Naive Bayes e SVM, utilizando sua configuração padrão da biblioteca Scikit-learn, ou seja, sem variar seus hiperparâmetros (exceto na Regressão Logística, em que utilizaremos um parâmetro para limitar o número de iterações e evitar que o código demore muito tempo para ser executado). Para uma melhor avaliação, utilizaremos o método de validação cruzada (10 *folds*) e compararemos os resultados graficamente através de *boxplots*.

O código está comentado, para facilitar o entendimento. Digite o trecho de código a seguir e veja o resultado:

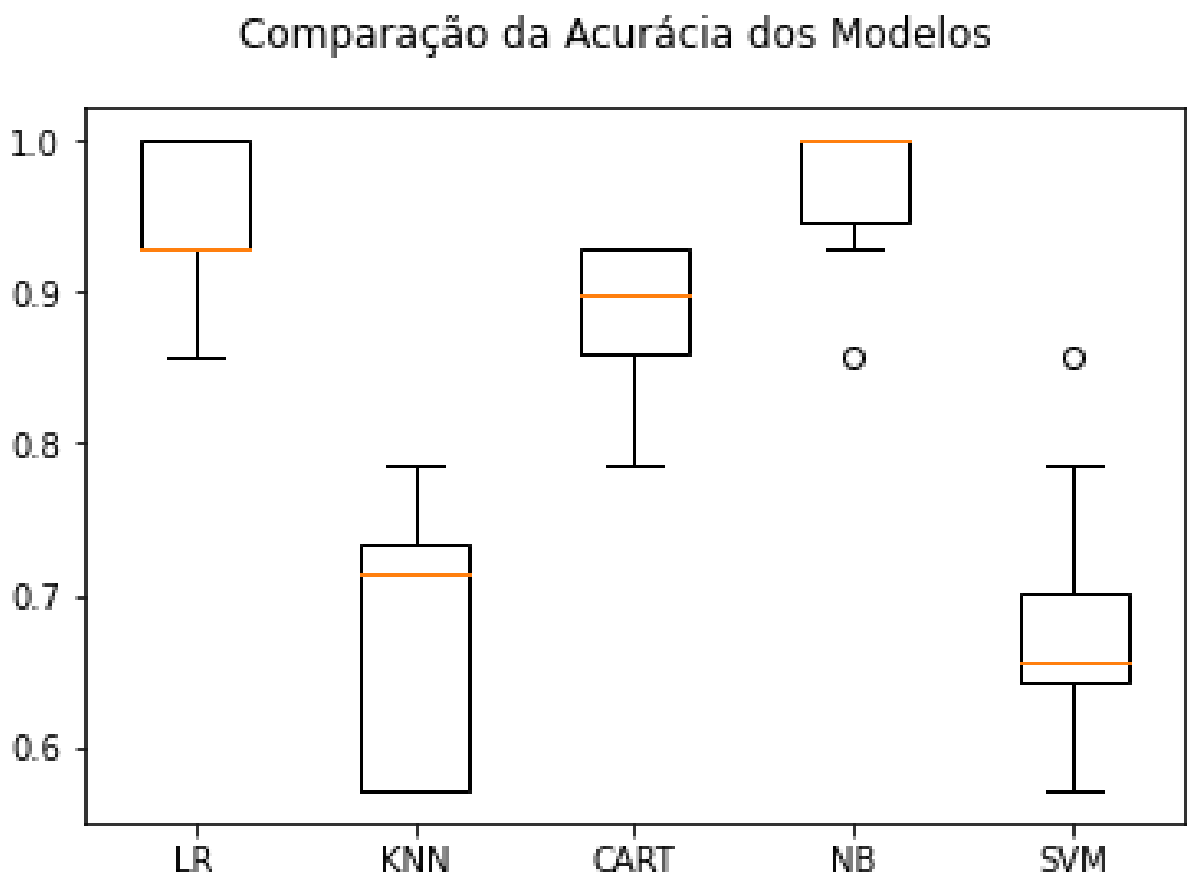
```
# Configuração para não exibir os warnings
import warnings
warnings.filterwarnings("ignore")

# Imports necessários
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine # para importar o dataset wine
from sklearn.model_selection import train_test_split # para particionar em
bases de treino e teste (holdout)
```

```
from sklearn.model_selection import KFold # para preparar os folds da validação cruzada
from sklearn.model_selection import cross_val_score # para executar a validação cruzada
from sklearn.metrics import accuracy_score # para a exibição da acurácia do modelo
from sklearn.linear_model import LogisticRegression # algoritmo Regressão Logística
from sklearn.neighbors import KNeighborsClassifier # algoritmo KNN
from sklearn.tree import DecisionTreeClassifier # algoritmo Árvore de Classificação
from sklearn.naive_bayes import GaussianNB # algoritmo Naive Bayes
from sklearn.svm import SVC # algoritmo SVM

# Preparação dos dados

# Carga do dataset
wine = load_wine()
dataset = pd.DataFrame(wine.data, columns=wine.feature_names) # conversão para dataframe
dataset['target'] = wine.target # adição da coluna target

# Separação em bases de treino e teste (holdout)
array = dataset.values
X = array[:,0:13] # atributos
y = array[:,13] # classe (target)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=7) # faz a divisão

# Definindo a métrica de avaliação dos algoritmos
scoring = 'accuracy'

# Criando os folds para a validação cruzada
num_particoes = 10 # número de folds da validação cruzada
kfold = KFold(n_splits=num_particoes, shuffle=True, random_state=7) # faz o particionamento em 10 folds

# Modelagem
```

```
# Definindo uma seed global para esta célula de código
np.random.seed(7)

# Listas para armazenar os modelos, os resultados e os nomes dos modelos
models = []
results = []
names = []

# Preparando os modelos e adicionando-os em uma lista
models.append(('LR', LogisticRegression(max_iter=200)))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))

# Avaliando um modelo por vez
for name, model in models:
  cv_results = cross_val_score(model, X_train, y_train, cv=kfold,
scoring=scoring)
  results.append(cv_results)
  names.append(name)
  msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std()) #
média e desvio padrão dos 10 resultados da validação cruzada
  print(msg)

# boxplot de comparação dos algoritmos
fig = plt.figure()
fig.suptitle('Comparação da Acurácia dos Modelos')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

A saída resultante será:

```
LR: 0.94 (0.05)
KNN: 0.68 (0.09)
CART: 0.88 (0.06)
NB: 0.97 (0.05)
SVM: 0.68 (0.08)
```



Analisando os resultados, verificamos que, considerando a acurácia média, o modelo treinado com o Naïve Bayes apresentou os melhores resultados (95% de acurácia média) seguido do modelo treinado com a Regressão Logística (94% de acurácia média), ambos com desvio padrão equivalente (5%). Já analisando os *boxplots*, vemos que a mediana da acurácia do modelo treinado com o Naïve Bayes é superior à do modelo treinado com a Regressão Logística, indicando que possivelmente seguiríamos com o Naïve Bayes como escolha de algoritmo. Neste caso, faríamos:

```
# Criando um modelo com todo o conjunto de treino
model = GaussianNB()
model.fit(X_train, y_train)

# Fazendo as predições com o conjunto de teste
predictions = model.predict(X_test)
```

```
# Estimando a acurácia no conjunto de teste
print(accuracy_score(y_test, predictions))
```

A saída resultante será:

```
1.0
```

Repare que conseguimos obter uma acurácia de teste de 100%. Reforçamos que este resultado **não é comum** quando trabalhamos com dados de problemas reais, mas possível com *datasets* simplificados similares a este, trabalhados e disponibilizados apenas para fins acadêmicos.

Ressaltamos que esse é um exemplo simples, apenas para ilustrar neste livro a aplicação de algoritmos de Machine Learning. Em problemas reais (com dados mais "sujos" e complexos), iríamos provavelmente trabalhar com algumas operações de pré-processamento de dados (tais como Normalização e Padronização, para dados quantitativos, e *One-Hot-Encoding*, para dados qualitativos nominais) e também experimentar variar os hiperparâmetros dos algoritmos, o que resultaria em um código mais complexo. Optamos por apresentar um exemplo simplificado, pois o foco deste livro é em conceitos estatísticos para Ciência de Dados, e não em Machine Learning. Se você tiver interesse em estudar sobre boas práticas da engenharia de software aplicadas a Machine Learning, bem como outros conceitos avançados de Machine Learning, recomendamos o livro *Engenharia de Software para Ciência de Dados: Um guia de boas práticas com ênfase na construção de sistemas de Machine Learning em Python*, de Marcos Kalinowski, Tatiana Escovedo, Hugo Villamizar e Hélio Lopes, que também é da Casa do Código.

Neste capítulo, apresentamos a segunda parte do tema Inferência Estatística, focando na parte de Modelagem Estatística e abordando os principais algoritmos de Machine Learning. No próximo capítulo, falaremos sobre Experimentação Contínua.

CAPÍTULO 8
Experimentação Contínua

A **Experimentação Contínua** (também conhecida como Teste A/B) é uma prática comum na área de Engenharia de Software e se concentra em avaliar continuamente novas versões de produtos de software, visando assegurar que estes produtos atendam a hipóteses de negócio preestabelecidas, permitindo a entrega de valor aos usuários de forma rápida e confiável.

8.1 Experimentação Contínua na Ciência de Dados

No contexto de Ciência de Dados, a Experimentação Contínua é uma prática importante para avaliar modelos de Machine Learning em relação ao alcance de seus objetivos, realizando testes de hipótese para subsidiar decisões referentes a sua implantação. A avaliação contínua proposta permite melhorar constantemente a qualidade e o desempenho dos modelos, tornando-os mais precisos, robustos e adequados para as tarefas em questão. Isso é fundamental em um ambiente em constante evolução, onde novos dados e desafios surgem regularmente. De forma resumida, a Experimentação Contínua pode ser utilizada em Machine Learning para avaliar hipóteses associadas a variações de modelos gerados através de diversas atividades vistas nos capítulos anteriores deste livro, tais como: ajuste de hiperparâmetros, seleção de características, variações de algoritmos de Machine Learning.

A Experimentação Contínua é aplicada neste contexto para testar hipóteses associadas a estes modelos, assegurando que um novo modelo melhore o desempenho do anterior com significância estatística. Estas hipóteses podem estar associadas tanto a métricas

específicas do modelo (como aumentar a acurácia ou o *recall* para problemas de classificação ou diminuir o RMSE para problemas de regressão), quanto a outros aspectos que podem afetar o uso do modelo em cenários operacionais (tais como o tempo de aprendizado, o tempo de inferência, entre outros) ou a resultados de negócio afetados pelo uso do modelo (como um aumento de vendas através de melhorias em uma classificação automática de produtos). Uma visão mais detalhada dos aspectos que podem precisar ser considerados encontra-se no livro *Engenharia de Software para Ciência de Dados* publicado pela editora Casa do Código.

Caso o novo modelo melhore o desempenho do anterior com significância estatística, confirmando as hipóteses associadas com as melhorias esperadas, ele deve ser implantado em produção e monitorado continuamente. Algumas empresas possuem *pipelines* automatizados para monitorar e retreinar modelos quando necessário, muitas vezes no contexto de soluções de MLOps (*Machine Learning Operations* – que visa automatizar o processo de levar modelos para produção, monitorar e manter esses modelos atualizados). Idealmente, a Experimentação Contínua deve fazer parte deste *pipeline* para assegurar que um novo modelo gerado realmente represente melhorias sobre o anterior e possa ser implantado.

Este capítulo apresentará conceitos e técnicas estatísticas que são utilizados na Experimentação Contínua no contexto de Machine Learning.

8.2 Hipótese

Uma **hipótese** é uma teoria ou suposição que é testada usando dados e análise estatística e que pode explicar um comportamento determinado do interesse de pesquisa. Por exemplo, considerando um modelo de Machine Learning para classificar bons e maus

pagadores de empréstimos bancários, uma hipótese a ser testada poderia ser: "O modelo de Machine Learning A sugere a concessão de crédito para clientes com melhor acurácia do que o modelo de Machine Learning B". Um estudo experimental tem como objetivo coletar dados, a partir de um ambiente controlado, para confirmar ou refutar a hipótese. Hipóteses orientam a definição de variáveis, a saber:

- **Variáveis independentes (tornam-se fatores quando controladas):** relacionam-se com as entradas do processo, podem ser controladas e representam as causas que se espera que afetem os resultados. Quando controladas, seus valores são chamados de tratamentos.
- **Variáveis dependentes:** relacionam-se com as saídas do processo e são afetadas ao longo do processo de experimentação. Representam o efeito da combinação dos valores das variáveis independentes (incluindo os fatores) e seus possíveis valores são chamados de resultados.

No exemplo "O modelo de Machine Learning A sugere a concessão de crédito para clientes com melhor acurácia do que o modelo de Machine Learning B", temos:

- Variáveis independentes: modelo utilizado e caracterização dos clientes.
- Variável dependente: acurácia.

No contexto da Experimentação Contínua, normalmente definimos dois tipos de hipótese:

- **Hipótese nula (H0)**: a hipótese nula é uma afirmação que indica a ausência de efeito, ou a ausência de diferença, ou um estado de neutralidade. A hipótese nula é o que normalmente um teste de hipóteses tenta rejeitar.
- **Hipótese alternativa (H1)**: a hipótese alternativa é uma afirmação que indica a presença de um efeito, uma diferença, ou uma relação entre variáveis. Ela é diretamente oposta à

hipótese nula e é o que esperamos que seja verdade ou o que estamos tentando provar. A hipótese alternativa é aceita se os dados fornecerem evidência suficiente para rejeitar a hipótese nula.

Voltando ao exemplo bancário, poderíamos listar as seguintes hipóteses:

- Hipótese nula (H0): o modelo de Machine Learning A sugere a concessão de crédito para clientes com a mesma acurácia que o modelo de Machine Learning B (ausência de efeito).
- Hipótese alternativa (H1): o modelo de Machine Learning A sugere a concessão de crédito para clientes com melhor acurácia do que o modelo de Machine Learning B (efeito esperado).

Os testes estatísticos permitem a aceitação ou rejeição das hipóteses, como veremos ainda neste capítulo.

8.3 Teste de hipótese

No contexto de Machine Learning, o **teste de hipótese** refere-se a um procedimento estatístico usado para avaliar se uma nova versão de um modelo é significativamente melhor do que uma versão de referência ou não. Esse teste é usado para tomar decisões sobre a implantação ou não de modelos de Machine Learning em produção de forma contínua. É aplicado na Experimentação Contínua em Machine Learning da seguinte forma:

1. Formulação das hipóteses: como vimos, existem dois tipos de hipóteses:
    - Hipótese nula (H0): assume que não há diferença significativa entre a versão atual do modelo (ou a versão de referência) e a nova versão que está sendo avaliada.

    - Hipótese alternativa (H1): assume que há uma diferença significativa entre a versão atual do modelo (ou a versão de referência) e a nova versão que está sendo avaliada.
2. Coleta de dados: predições de ambos os modelos são coletadas para diversos conjuntos de teste para medir o desempenho nas métricas de interesse para cada um destes conjuntos. Essas métricas devem refletir as variáveis dependentes do experimento e podem ser, por exemplo, a acurácia, a precisão ou o erro médio quadrático.

3. Análise Estatística: os resultados obtidos com os dois modelos são comparados usando técnicas estatísticas apropriadas. Isso pode envolver testes de hipótese estatísticos, como o teste *t de Student*, testes de *Mann-Whitney*, ou outros testes adequados ao problema (que veremos com mais detalhes ainda neste capítulo).

4. Aceitar ou rejeitar a hipótese nula: com base na análise estatística, a hipótese nula é aceita ou rejeitada:

    - Aceitar a hipótese nula: significa que não há evidência estatística de que a nova versão do modelo seja significativamente diferente da versão atual (ou de referência), e a nova versão não é implantada.
    - Rejeitar a hipótese nula: significa que há evidência estatística de que a nova versão do modelo é significativamente melhor ou pior do que a versão atual (ou de referência). Nesse caso, a nova versão pode ser implantada ou não, dependendo da direção da diferença e dos objetivos do experimento.

Portanto, no contexto de Experimentação Contínua em Machine Learning, o teste de hipótese é uma ferramenta estatística usada para avaliar e tomar decisões sobre a implantação de modelos de Machine Learning em produção, garantindo que essas mudanças tragam melhorias.

Tipos de testes de hipótese

Como vimos, um estudo experimental tem como objetivo coletar dados para confirmar ou refutar uma hipótese. Dentro da etapa de Análise Estatística, a escolha do teste estatístico apropriado depende da natureza dos dados e do tipo de comparação que está sendo realizada. Existem vários tipos de testes de hipótese que podem ser aplicados, dependendo das circunstâncias. Os tipos de testes de hipótese mais comumente utilizados nesse contexto são:

Teste t de Student:

- Uso: Comparação de médias entre dois grupos de dados quando os dados seguem uma distribuição normal.
- Exemplo: Comparação da acurácia média de dois modelos de Machine Learning, onde a acurácia de cada um dos modelos segue uma distribuição normal.

Teste de Análise de Variância (ANOVA):

- Uso: Comparação de médias entre três ou mais grupos quando os dados seguem uma distribuição normal.
- Exemplo: Comparação da acurácia média de vários modelos de Machine Learning, onde a acurácia de cada um dos modelos segue uma distribuição normal.

Teste de Mann-Whitney:

- Uso: Comparação de distribuições entre dois grupos quando os dados não seguem uma distribuição normal.
- Exemplo: Comparação da acurácia média de dois modelos de Machine Learning, onde a acurácia de cada um dos modelos não segue uma distribuição normal.

Teste de Kruskal-Wallis:

- Uso: Comparação das medianas de três ou mais grupos independentes quando dados não seguem uma distribuição

normal. É uma alternativa não paramétrica ao teste ANOVA.
- Exemplo: Comparação da acurácia média de vários modelos de Machine Learning, onde a acurácia de cada um dos modelos não segue uma distribuição normal.

Os testes de hipótese podem ser **paramétricos** ou **não paramétricos**. Os testes paramétricos usam fórmulas específicas derivadas de frequências de distribuição conhecidas, significando que o conjunto de dados que será testado deve apresentar uma distribuição normal (distribuição simétrica) e homocedástica (a variância dos dados é constante em todas as categorias ou grupos em estudo). Já os testes não paramétricos devem ser usados quando a distribuição dos dados não atende aos requisitos dos testes paramétricos (normalidade e homocedasticidade) ou quando não é possível determinar a distribuição subjacente dos dados. Eles são menos poderosos do que os testes paramétricos, mas não pressupõem nenhuma distribuição de probabilidade nos dados.

Ainda neste capítulo veremos como testar a normalidade e a homocedasticidade de uma distribuição de dados. A figura a seguir resume os tipos de testes de hipótese paramétricos e não paramétricos que abordaremos neste capítulo:

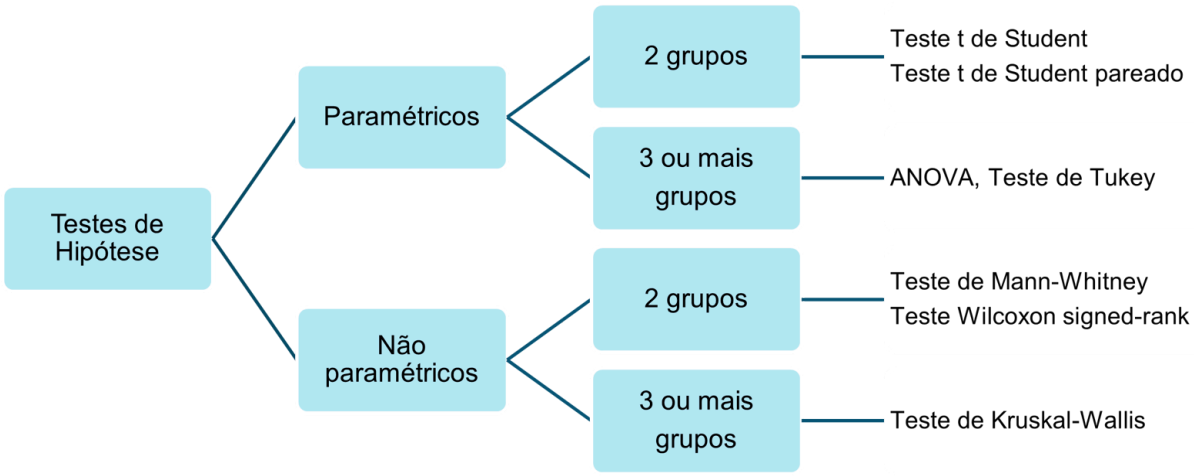


Figura 8.1: Tipos de testes de hipótese.

Como mencionado, a escolha do teste de hipótese depende da natureza dos dados, do tamanho da amostra, da distribuição dos dados e da pergunta específica que está sendo investigada. Se os dados se aproximarem de uma distribuição normal e atenderem às outras suposições dos testes paramétricos, esses testes geralmente são preferíveis devido ao seu maior poder estatístico. Como veremos a seguir, também é importante definir o nível de significância (alpha) a ser testado para controlar o risco de erro tipo I (rejeitar erroneamente a hipótese nula) e interpretar os resultados com cuidado. Experimentação Contínua em Machine Learning envolve a coleta e análise contínua de dados, tornando a escolha e aplicação dos testes de hipótese uma parte crítica do processo de avaliação e tomada de decisões. Também neste capítulo veremos mais detalhes e exemplos de códigos destes testes de hipótese.

Tipos de erros

A verificação de hipóteses sempre envolve algum risco, implicando que alguns erros de análise podem ocorrer:

- **Erro Tipo I (α)**: ocorre quando o teste estatístico indica a existência de uma relação entre causa e efeito que na verdade não existe.
- **Erro Tipo II (β)**: ocorre quando o teste estatístico não indica uma relação entre causa e efeito que na verdade existe.

Ou seja:

- α = P (erro tipo I) = P (H0 é rejeitada | H0 é verdadeira)
- β = P (erro tipo II) = P (H0 não é rejeitada | H0 é falsa)

A hipótese nula geralmente é construída para evitar erros do tipo I. Por exemplo, considere a acurácia de dois modelos de Machine Learning:

- H0: modelo A = modelo B
- H1: modelo A é melhor do que o modelo B

Nesse caso, os possíveis erros são:

- Tipo I: o teste indica que o modelo A é melhor do que B, mas isso não é verdade (eles são iguais)
- Tipo II: o teste indica que o modelo A é igual ao modelo B, mas isso não é verdade (A é melhor)

Nível de significância

O **nível de significância** (também conhecido como α) apresenta a probabilidade de ocorrer um erro tipo I: quanto menor o α escolhido, menor a probabilidade de cometer um erro tipo I. Os níveis de significância mais comuns são 10%, 5%, 1% e 0,1% e representam o limite além do qual consideramos um resultado estatisticamente significativo. Por exemplo, se escolhermos um α de 5%, estamos dispostos a aceitar no máximo uma chance de 5% de cometer um erro tipo I ao rejeitar a hipótese nula.

Chamamos de **p-valor** (em inglês, *p-value*) o nível de significância mais baixo que pode ser usado para rejeitar a hipótese nula. O p-valor é uma medida estatística que indica a probabilidade de obter os resultados observados sob a suposição de que a hipótese nula seja verdadeira. Quanto menor o p-valor obtido pelo teste, menos provável é que os resultados tenham ocorrido ao acaso e mais evidência temos contra a hipótese nula.

Dizemos que existe **significância estatística** quando o p-valor obtido é menor do que o nível de significância (α) adotado. Isso significa que os resultados são tão extremos que são improváveis de ocorrer sob a suposição de que a hipótese nula seja verdadeira. Portanto, podemos rejeitar a hipótese nula em favor da hipótese alternativa.

Assim, se o p-valor for menor ou igual a um nível de significância pré-determinado (α), rejeitamos a hipótese nula e afirmamos apoio à hipótese alternativa. Se o p-valor for maior que o nível α, não

rejeitamos a hipótese nula e não podemos afirmar apoio à hipótese alternativa. Vejamos alguns exemplos:

- Quando p = 0,0001 (um valor baixo) e α = 5% (0,05), pode-se claramente dizer que o resultado é significativo, pois o p-valor é muito menor do que α. Um p-valor muito baixo sugere que os resultados são altamente improváveis de ocorrer ao acaso, tornando a evidência contra a hipótese nula muito forte.
- No entanto, se p = 0,048 e α = 5% (0,05), embora o p-valor seja menor do que α, os valores estão muito próximos. Neste caso, a hipótese nula pode ser rejeitada para α = 5%, mas a evidência contra a hipótese nula é menos convincente.

8.4 Teste de normalidade

Como vimos anteriormente, é necessário conhecer a distribuição dos dados a fim de escolher qual tipo de teste de hipótese utilizar. Os testes paramétricos só devem ser utilizados em amostras que seguem uma distribuição normal. Para tal, o teste de normalidade é um procedimento estatístico usado para verificar se um conjunto de dados segue ou se aproxima de uma distribuição normal. Já vimos em capítulos anteriores que a distribuição normal é uma das distribuições de probabilidade mais comuns e é caracterizada por sua forma de sino, simetria e propriedades matemáticas bem definidas. Assim, a realização de um teste de normalidade é importante em casos que se pressupõe que os dados tenham uma distribuição normal. Neste contexto, dois testes de normalidade comumente usados são o **Teste de Kolmogorov-Smirnov** e o **Teste de Shapiro-Wilk**.

De forma simplificada, a interpretação do resultado destes testes é feita comparando o p-valor com um nível de significância escolhido. Se o p-valor for maior do que o nível de significância, não rejeitamos

a hipótese nula e concluímos que os dados parecem seguir uma distribuição normal. Se o p-valor for menor do que o nível de significância, rejeitamos a hipótese nula e concluímos que os dados não seguem uma distribuição normal. Veremos a seguir mais detalhes e exemplos destes dois testes.

Teste de Kolmogorov-Smirnov (K-S)

O **Teste de Kolmogorov-Smirnov (K-S)** é um teste de normalidade utilizado para verificar se uma amostra de dados segue uma distribuição de probabilidade específica, como uma distribuição normal ou outra distribuição teórica. Esse teste também pode ser usado para comparar duas amostras e verificar se elas foram retiradas de populações com a mesma distribuição. Ele compara a distribuição empírica dos dados (observada) com a distribuição teórica especificada. Esse teste é frequentemente usado para identificar normalidade em amostras com pelo menos 30 valores e detecta diferenças relacionadas à tendência central, dispersão e simetria.

O Teste de Kolmogorov-Smirnov é baseado na ideia de que, se os dados seguirem a distribuição teórica especificada, a função de distribuição acumulada (CDF) empírica dos dados deve ser próxima à CDF da distribuição teórica. A estatística de teste do K-S quantifica a maior discrepância entre as duas funções de distribuição acumulada. Esse teste pode ser afetado pelo tamanho da amostra, tornando-se mais poderoso com amostras maiores, que contenham ao menos 30 elementos.

Vejamos um exemplo simples de como realizar o Teste de Kolmogorov-Smirnov em Python usando a biblioteca `scipy` para testar se uma amostra de dados segue uma distribuição normal:

```
from scipy import stats
import numpy as np

# Amostra de dados que queremos testar (50 elementos: note que este teste
```

```
# é recomendado para amostras grandes, com pelo menos 30 elementos)
amostra = [0.81, 0.85, 0.76, 0.91, 0.94, 0.90, 0.70, 0.83, 0.92, 0.84,
0.72,
           0.85, 0.70, 0.75, 0.93, 0.86, 0.94, 0.82, 0.77, 0.83, 0.88,
0.83,
           0.93, 0.80, 0.80, 0.77, 0.67, 0.80, 0.85, 0.83, 0.88, 0.81,
0.83,
           0.82, 0.78, 0.79, 0.87, 0.83, 0.84, 0.85, 0.84, 0.77, 0.67,
0.87,
           0.94, 0.80, 0.70, 0.75, 0.71, 0.78]

# Realizando o Teste de Kolmogorov-Smirnov
ks_statistic, p_value = stats.kstest(amostra, 'norm')

print(f"Estatística do teste KS: {ks_statistic}")
print(f"p-valor: {p_value}")

# Interpretando o resultado
alpha = 0.05 # Nível de signficância escolhido
if p_value > alpha:
    print("A distribuição é normal, pois de acordo com o teste, não há
evidências suficientes para rejeitar a hipótese de normalidade.")
else:
    print("A distribuição não pode ser considerada normal, pois de acordo
com o teste, rejeita-se a hipótese de normalidade.")
```

A saída resultante será:

```
Estatística do teste KS: 0.7485711049046899
p-valor: 1.8167127952549e-29
A distribuição não pode ser considerada normal, pois de acordo com o
teste, rejeita-se a hipótese de normalidade.
```

Observe que a interpretação do resultado é baseada no p-valor: se o p-valor for maior que o nível de significância escolhido (no caso do código de exemplo, 0,05), não rejeitemos a hipótese nula (H0), o que significa que os dados parecem seguir uma distribuição normal. Caso contrário, se o p-valor for menor que o nível de significância

escolhido, rejeitaremos H0, indicando que os dados não seguem uma distribuição normal.

Teste de Shapiro-Wilk

O **Teste de Shapiro-Wilk** é um teste de normalidade estatístico usado para determinar se uma amostra de dados segue uma distribuição normal. Ele é amplamente utilizado para avaliar a normalidade dos dados em análises estatísticas, pois é sensível a desvios da normalidade, especialmente em amostras de tamanho pequeno a moderado. Por isso, é frequentemente usado para identificar normalidade em amostras com menos de 50 valores. Assim como no teste anterior, a hipótese nula (H0) do Teste de Shapiro-Wilk afirma que os dados foram amostrados de uma população que segue uma distribuição normal. A hipótese alternativa (H1) é que os dados não foram amostrados de uma população com uma distribuição normal.

O Teste de Shapiro-Wilk calcula uma estatística de teste (S-W) com base nas diferenças entre os valores observados e os valores esperados sob a hipótese de normalidade. Esta estatística de teste é usada para calcular o p-valor. Apesar de ser adequado para amostra menores, o Teste de Shapiro-Wilk pode também não ser apropriado para amostras muito pequenas (por exemplo, com 3 a 5 observações) e pode ser influenciado por *outliers* (valores extremos). Além disso, ele é mais apropriado para dados contínuos, não sendo recomendado para dados categóricos.

Vejamos um exemplo simples de como realizar o Teste de Shapiro-Wilk em Python usando a biblioteca `scipy` para testar se uma amostra de dados segue uma distribuição normal:

```
from scipy import stats
import numpy as np

# Amostra de dados que queremos testar (30 elementos: note que este teste
# é recomendado para amostras menores, com menos de 50 elementos)
```

```
amostra = [0.71, 0.84, 0.89, 0.67, 0.90, 0.79, 0.71, 0.74, 0.71, 0.78,
           0.75, 0.77, 0.80, 0.92, 0.72, 0.69, 0.66, 0.75, 0.86, 0.76,
           0.68, 0.78, 0.87, 0.86, 0.82, 0.74, 0.87, 0.81, 0.95, 0.69]

# Realizando o Teste de Shapiro-Wilk
shapiro_statistic, p_value = stats.shapiro(amostra)

print(f"Estatística do teste de Shapiro-Wilk (S-W): {shapiro_statistic}")
print(f"p-valor: {p_value}")

# Interpretando o resultado
alpha = 0.05 # Nível de signficância escolhido
if p_value > alpha:
    print("A distribuição é normal, pois de acordo com o teste, não há
evidências suficientes para rejeitar a hipótese de normalidade.")
else:
    print("A distribuição não pode ser considerada normal, pois de acordo
com o teste, rejeita-se a hipótese de normalidade.")
```

A saída resultante será:

```
Estatística do teste de Shapiro-Wilk (S-W): 0.9606984257698059
p-valor: 0.32273441553115845
A distribuição é normal, pois de acordo com o teste, não há evidências
suficientes para rejeitar a hipótese de normalidade.
```

Assim como no exemplo anterior, a interpretação do resultado é baseada no p-valor. Se o p-valor for maior que o nível de significância escolhido (no caso do código de exemplo, 0,05), não rejeitaremos a hipótese nula (H0), o que significa que os dados parecem seguir uma distribuição normal. Caso contrário, se o p-valor for menor que o nível de significância escolhido, rejeitaremos H0, indicando que os dados não seguem uma distribuição normal.

Esses exemplos ilustram como verificar a normalidade dos dados antes de prosseguir com testes que assumem essa condição. A escolha entre o Teste de Kolmogorov-Smirnov e o Teste de Shapiro-Wilk pode depender do tamanho da sua amostra e do objetivo

específico da sua análise. Em geral, o Teste de Shapiro-Wilk é preferido para amostras menores.

8.5 Homocedasticidade

Como vimos anteriormente, a **homocedasticidade** é um pressuposto crucial para a aplicação de testes paramétricos, que exigem que as variâncias das observações em diferentes grupos ou categorias sejam próximas. Isso significa que a variabilidade dos dados em torno da média deve ser semelhante entre os grupos comparados. A homocedasticidade garante que as inferências estatísticas feitas a partir dos testes sejam válidas, uma vez que a igualdade das variâncias contribui para a precisão das estimativas de efeitos e das comparações entre grupos.

Em Machine Learning, considerar a homocedasticidade é relevante, por exemplo, ao comparar a distribuição das acurácias de dois modelos de aprendizado de máquina. A falta de homocedasticidade pode indicar que os resultados de um modelo podem ser mais inconsistentes ou variáveis em relação ao outro, o que pode afetar a interpretação dos resultados.

Teste de homocedasticidade: Teste de Levene

Para verificar a homocedasticidade, o **Teste de Levene** é frequentemente utilizado. Esse teste avalia se as variâncias entre grupos são homogêneas. O teste tem duas hipóteses:

- Hipótese nula (H0): não há diferença significativa nas variâncias entre os grupos.
- Hipótese alternativa (H1): pelo menos um grupo tem uma variância significativamente diferente dos outros.

O resultado do Teste de Levene nos ajuda a decidir se as variâncias são homogêneas (homocedásticas) e, portanto, se é apropriado

prosseguir com testes paramétricos que assumem essa condição.

Para ilustrar o Teste de Levene no contexto de Machine Learning, vamos considerar um cenário em que temos dois modelos de Machine Learning, Modelo A e Modelo B, e queremos verificar se a variabilidade (variância) da acurácia dos modelos é consistente entre os dois. Usaremos o Teste de Levene para determinar se as acurácias dos modelos têm variâncias homogêneas ou não. Vejamos um exemplo simples de como realizar o Teste de Levene em Python neste contexto usando a biblioteca `scipy`:

```
from scipy.stats import levene
import numpy as np

# Acurácias simuladas do Modelo A e Modelo B
acuracias_modelo_a = np.array([0.82, 0.81, 0.83, 0.84, 0.82, 0.80, 0.81,
0.83, 0.85, 0.82])
acuracias_modelo_b = np.array([0.76, 0.77, 0.75, 0.78, 0.76, 0.79, 0.75,
0.77, 0.74, 0.76])

# Realizando o Teste de Levene para avaliar a homocedasticidade
statistic, p_value = levene(acuracias_modelo_a, acuracias_modelo_b)

print(f"Estatística do teste: {statistic}")
print(f"p-valor: {p_value}")

# Interpretando o resultado
alpha = 0.05 # Nível de signficância escolhido
if p_value > alpha:
    print("As variâncias são homogêneas. Não há evidências suficientes
para rejeitar a hipótese nula.")
else:
    print("As variâncias não são homogêneas. Rejeita-se a hipótese nula,
indicando diferenças significativas nas variâncias entre os grupos.")
```

A saída resultante será:

```
Estatística do teste: 5.507970293902395e-29
p-valor: 1.0
```

```
As variâncias são homogêneas. Não há evidências suficientes para rejeitar a hipótese nula.
```

Este exemplo ilustra como o Teste de Levene pode ser aplicado em um cenário de Machine Learning para verificar a homocedasticidade das acurácias de modelos diferentes. Note que, se o p-valor for maior que 0.05 (o nível de significância escolhido neste exemplo de código), concluímos que as variâncias são homogêneas. Isso implica que a variabilidade da acurácia entre os dois modelos não difere significativamente, e os testes estatísticos subsequentes que assumem homocedasticidade podem ser aplicados com confiança. Se o p-valor for menor ou igual a 0.05, não há evidência para assumir que as variâncias sejam homogêneas, o que pode exigir ajustes nas análises ou a escolha de métodos estatísticos que não assumam a homocedasticidade.

8.6 Testes de hipótese paramétricos

Teste t de Student (ou Student-t)

O **Teste t de Student**, também conhecido como **Teste de hipótese Student-t**, é utilizado para comparar as médias de duas amostras independentes, verificando se há diferenças estatísticas significativas entre elas. Um dos exemplos de aplicação deste teste seria analisar se dois modelos de Machine Learning produzem resultados diferentes em termos de acurácia com significância estatística, como ilustra o exemplo a seguir:

```
from scipy import stats

# Acurácias de dois modelos de Machine Learning em uma tarefa específica
acuracias_modelo_1 = [0.82, 0.80, 0.81, 0.79, 0.83]
acuracias_modelo_2 = [0.76, 0.75, 0.77, 0.74, 0.78]

# Aplicando o teste t de Student
```

```
t_statistic, p_value = stats.ttest_ind(acuracias_modelo_1,
acuracias_modelo_2)

# Exibindo os resultados do teste
print("Estatística de teste:", t_statistic)
print("p-valor:", p_value)

# Interpretando o resultado
alpha = 0.05  # Nível de significância escolhido
if p_value < alpha:
    print("Rejeita a hipótese nula: as acurácias são estatisticamente
diferentes")
else:
    print("Não há evidência suficiente para rejeitar a hipótese nula: as
acurácias são estatisticamente iguais")
```

A saída resultante será:

```
Estatística de teste: 4.999999999999986
p-valor: 0.0010528257933665572
Rejeita a hipótese nula: as acurácias são estatisticamente diferentes
```

É importante destacar que, no contexto da Experimentação Contínua referente a modelos de Machine Learning, podemos ter duas situações: testando a acurácia de modelos com conjuntos de dados de teste diferentes ou com conjuntos de dados de teste iguais. Se os conjuntos de dados de teste forem os mesmos, estamos comparando resultados relacionados (ou seja, podemos interpretar os resultados obtidos para um mesmo conjunto de dados em cada um dos modelos como resultados relacionados). Neste caso, o **Teste t de Student pareado** (utilizando o comando `stats.ttest_rel` em vez de `stats.ttest_ind`) é mais indicado por permitir análises mais precisas para observações dependentes ou pareadas, tendo maior sensibilidade para detectar diferenças estatisticamente significativas nesta situação.

Teste de análise de variância (ANOVA)

O **Teste ANOVA** (Análise de Variância) é aplicado para comparar as médias de três ou mais grupos independentes. No contexto de Machine Learning, pode ser usado, por exemplo, para avaliar se diferentes configurações de um modelo resultam em acurácias significativamente distintas. Vejamos o exemplo a seguir:

```
from scipy import stats

# Acurácias de três modelos de Machine Learning
acuracias_modelo_a = [0.82, 0.81, 0.83, 0.84, 0.82]
acuracias_modelo_b = [0.75, 0.77, 0.76, 0.74, 0.75]
acuracias_modelo_c = [0.89, 0.90, 0.91, 0.88, 0.87]

# Aplicando ANOVA
f_statistic, p_value = stats.f_oneway(acuracias_modelo_a,
acuracias_modelo_b, acuracias_modelo_c)

# Exibindo os resultados do teste
print("Estatística de teste:", f_statistic)
print("p-valor:", p_value)

# Interpretando o resultado
alpha = 0.05  # Nível de significância escolhido
if p_value < alpha:
    print("Rejeita a hipótese nula: as acurácias são estatisticamente
diferentes")
else:
    print("Não há evidência suficiente para rejeitar a hipótese nula: as
acurácias são estatisticamente iguais")
```

A saída resultante será:

```
Estatística de teste: 136.03921568627453
p-valor: 5.6814332556353306e-09
Rejeita a hipótese nula: as acurácias são estatisticamente diferentes
```

Teste de Tukey

Após identificar que há diferenças significativas com o Teste ANOVA, o **Teste de Tukey** ajuda a determinar quais pares específicos de grupos diferem entre si, considerando um nível de significância predefinido (geralmente α = 0,05). Esse teste é útil, por exemplo, para comparar as acurácias de vários modelos de Machine Learning em detalhes. Vejamos o exemplo a seguir:

```
from statsmodels.stats.multicomp import pairwise_tukeyhsd
import numpy as np

# Acurácias de três modelos de Machine Learning e seus respectivos grupos
acuracias = np.array([0.82, 0.81, 0.83, 0.84, 0.82, 0.75, 0.77, 0.76,
0.74, 0.75, 0.89, 0.90, 0.91, 0.88, 0.87])
# Rótulos para identificar as acurácias de cada modelo
grupos = ['Modelo A'] * 5 + ['Modelo B'] * 5 + ['Modelo C'] * 5

# Aplicando o Teste de Tukey
tukey_resultados = pairwise_tukeyhsd(acuracias, grupos)

# Exibindo os resultados do teste
print(tukey_resultados)
```

A saída resultante será:

```
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1   group2  meandiff p-adj lower  upper  reject
-----------------------------------------------------
Modelo A Modelo B    -0.07   0.0 -0.092 -0.048   True
Modelo A Modelo C    0.066   0.0  0.044  0.088   True
Modelo B Modelo C    0.136   0.0  0.114  0.158   True
-----------------------------------------------------
```

Aqui está o que cada parte dos resultados do Teste de Tukey significa:

- Grupos comparados: mostra os pares de grupos que estão sendo comparados.

- Diferença de média (`meandiff`): a diferença entre as médias dos dois grupos comparados. Um valor positivo indica que a média do primeiro grupo listado é maior que a do segundo grupo.
- Limite inferior e superior (`lower` , `upper`): o intervalo de confiança para a diferença entre as médias dos grupos. Se este intervalo não incluir 0, a diferença é considerada estatisticamente significativa.
- P-valor (`p-adj`): fornece a significância estatística da diferença observada entre os grupos. Um p-valor menor que o nível de significância α (por exemplo, 0,05) indica que há uma diferença estatisticamente significativa entre as médias dos grupos comparados. Isso significa que é improvável que a diferença observada tenha ocorrido por acaso.
- Rejeição da hipótese nula (`reject`): uma indicação de se a hipótese nula de não haver diferença entre os grupos foi rejeitada ou não, baseada no p-valor. "True" significa que a hipótese nula foi rejeitada (indicando diferenças significativas), e "False" significa que a hipótese nula não foi rejeitada (indicando que as diferenças não são significativas).

No contexto de Machine Learning, se você está comparando a acurácia de vários modelos e o Teste de Tukey mostra uma diferença significativa entre dois modelos (por exemplo, Modelo A e Modelo B), isso indica que a diferença nas acurácias dos modelos é estatisticamente significativa e provavelmente não ocorreu por acaso. Isso pode ajudar na decisão de qual modelo tem melhor desempenho geral ou em quais condições específicas um modelo pode ser preferido em relação a outro.

Já neste segundo exemplo do Teste de Tukey, as acurácias dos três modelos de Machine Learning (Modelo X, Modelo Y e Modelo Z) foram comparadas, e os resultados indicam que as diferenças entre os grupos não são estatisticamente significativas:

```
from statsmodels.stats.multicomp import pairwise_tukeyhsd
import numpy as np
```

```
# Acurácias de três modelos de Machine Learning com diferenças não significativas
# Simulando dados onde as acurácias dos modelos são mais próximas entre si
acuracias_similares = np.array([0.82, 0.83, 0.81, 0.82, 0.83, 0.81, 0.82, 0.83, 0.81, 0.82,
                                0.82, 0.83, 0.81, 0.82, 0.83])
# Rótulos para identificar as acurácias de cada modelo
grupos_similares = ['Modelo X'] * 5 + ['Modelo Y'] * 5 + ['Modelo Z'] * 5

# Aplicando o Teste de Tukey
tukey_resultados_similares = pairwise_tukeyhsd(acuracias_similares, grupos_similares)

print(tukey_resultados_similares)
```

A saída resultante será:

```
  Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1   group2  meandiff p-adj   lower  upper  reject
-----------------------------------------------------
Modelo X Modelo Y   -0.004 0.7359 -0.0181 0.0101  False
Modelo X Modelo Z      0.0    1.0 -0.0141 0.0141  False
Modelo Y Modelo Z    0.004 0.7359 -0.0101 0.0181  False
-----------------------------------------------------
```

Isso é evidenciado pelos seguintes aspectos dos resultados:

- Diferença de média (`meandiff`): as diferenças entre as médias dos grupos são muito pequenas (-0.004, 0.0, e 0.004), indicando que as acurácias dos modelos são muito próximas entre si.
- P-valor (`p-adj`): todos os p-valores são muito maiores que o nível de significância comum de 0.05 (0.7256, 0.9, e 0.7256). Isso sugere que não há evidências suficientes para rejeitar a hipótese nula de igualdade de médias entre os grupos.
- Rejeição da hipótese nula (`reject`): a coluna "reject" mostra "False" para todas as comparações, o que significa que a hipótese nula de que não há diferenças significativas entre as

médias dos grupos não foi rejeitada para nenhum par de grupos.

Portanto, neste caso, com base nos resultados do Teste de Tukey, podemos concluir que não há diferenças significativas nas acurácias dos modelos X, Y e Z. Isso implica que, para os dados e o contexto específicos deste exemplo, a escolha entre esses modelos pode ser baseada em critérios diferentes da acurácia, já que estatisticamente, eles apresentam desempenho similar.

8.7 Testes de hipótese não paramétricos

Teste de Mann-Whitney

O **Teste de Mann-Whitney** (também conhecido como **U de Mann-Whitney**, **Wilcoxon-Mann-Whitney** e **Wilcoxon Rank-Sum**) é utilizado para comparar duas amostras independentes, especialmente quando não se pode assumir a normalidade dos dados. No contexto de Machine Learning, ele pode ser usado, por exemplo, para comparar a acurácia de dois modelos em conjuntos de dados que não são normalmente distribuídos. Vejamos um exemplo:

```
from scipy.stats import mannwhitneyu
import numpy as np

# Acurácias simuladas do Modelo 1 e Modelo 2
acuracias_modelo_1 = np.random.beta(a=2.5, b=5, size=30) * 100  #
Multiplicado por 100 para representar porcentagem
acuracias_modelo_2 = np.random.beta(a=2, b=5, size=30) * 100  #
Multiplicado por 100 para representar porcentagem

# Aplicando o Teste de Mann-Whitney
u_statistic, p_value = mannwhitneyu(acuracias_modelo_1,
acuracias_modelo_2, alternative='two-sided')
```

```
# Exibindo os resultados do teste
print("Estatística de teste:", u_statistic)
print("p-valor:", p_value)

# Interpretando o resultado
alpha = 0.05  # Nível de significância escolhido
if p_value < alpha:
    print("Rejeita a hipótese nula: as acurácias são estatisticamente
diferentes")
else:
    print("Não há evidência suficiente para rejeitar a hipótese nula: as
acurácias são estatisticamente iguais")
```

A saída resultante será:

```
Estatística de teste: 510.0
p-valor: 0.37903631077393585
Não há evidência suficiente para rejeitar a hipótese nula: as acurácias
são estatisticamente iguais
```

Novamente, se os conjuntos de dados de teste para os modelos forem os mesmos, podemos considerar que estamos comparando resultados relacionados. Neste caso, a alternativa pareada para este teste não paramétrico, o **Teste Wilcoxon Signed-Rank** (utilizando o comando `stats.wilcoxon` ao invés de `stats.mannwhitneyu`) é mais indicado por permitir análises mais precisas para observações dependentes ou pareadas, tendo maior sensibilidade para detectar diferenças estatisticamente significativas nesta situação.

Teste de Kruskal-Wallis

O **Teste de Kruskal-Wallis**, também conhecido como **ANOVA não paramétrica**, é útil para comparar três ou mais grupos independentes sob a condição de não normalidade dos dados. Pode ser aplicado, por exemplo, para avaliar se diferentes configurações de um algoritmo de Machine Learning produzem acurácias significativamente diferentes. Vejamos um exemplo:

```
from scipy.stats import kruskal
import numpy as np

# Acurácias simuladas dos Modelos A, B e C
acuracias_modelo_a = [0.82, 0.93, 0.77, 0.76, 0.77, 0.75, 0.71, 0.78,
0.84, 0.79,
                      0.81, 0.81, 0.66, 0.94, 0.88, 0.94, 0.94, 0.9, 0.81,
0.65,
                      0.73, 0.67, 0.91, 0.89, 0.74, 0.82, 0.7, 0.83, 0.92,
0.84,
                      0.83, 0.87, 0.88, 0.84, 0.84, 0.94, 0.78, 0.75,
0.88, 0.89]
acuracias_modelo_b = [0.73, 0.69, 0.72, 0.79, 0.84, 0.73, 0.72, 0.68,
0.92, 0.75,
                      0.68, 0.73, 0.87, 0.67, 0.9, 0.94, 0.88, 0.88, 0.92,
0.8,
                      0.82, 0.87, 0.8, 0.87, 0.89, 0.72, 0.68, 0.82, 0.72,
0.68,
                      0.73, 0.83, 0.73, 0.88, 0.89, 0.94, 0.71, 0.9, 0.74,
0.9]
acuracias_modelo_c = [0.7, 0.87, 0.78, 0.84, 0.73, 0.78, 0.89, 0.93, 0.86,
0.85,
                      0.92, 0.85, 0.68, 0.93, 0.94, 0.74, 0.82, 0.81,
0.72, 0.84,
                      0.85, 0.8, 0.71, 0.86, 0.9, 0.7, 0.65, 0.93, 0.8,
0.73,
                      0.73, 0.76, 0.93, 0.7, 0.79, 0.87, 0.87, 0.86, 0.83,
0.91]

# Aplicando o Teste de Kruskal-Wallis para avaliar diferenças entre as
acurácias dos modelos
h_statistic, p_value = kruskal(acuracias_modelo_a, acuracias_modelo_b,
acuracias_modelo_c)

# Exibindo os resultados do teste
print("Estatística de teste:", h_statistic)
print("p-valor:", p_value)

# Interpretando o resultado
```

```
alpha = 0.05  # Nível de significância escolhido
if p_value < alpha:
    print("Rejeita a hipótese nula: as acurácias são estatisticamente diferentes")
else:
    print("Não há evidência suficiente para rejeitar a hipótese nula: as acurácias são estatisticamente iguais")
```

A saída resultante será:

```
Estatística de teste: 1.3169266267289885
p-valor: 0.5176461835905364
Não há evidência suficiente para rejeitar a hipótese nula: as acurácias são estatisticamente iguais
```

8.8 Tamanho do efeito (effect size)

O **tamanho do efeito (*effect size*)** é uma medida estatística que quantifica a magnitude de uma diferença ou relação entre duas variáveis ou grupos em um estudo. Diferentemente dos testes de hipótese, que apenas indicam se uma diferença é estatisticamente significativa, o tamanho do efeito ajuda a entender quão grande é essa diferença em termos práticos. Isso é especialmente útil em Machine Learning para comparar o desempenho de diferentes modelos ou configurações, indo além da mera significância estatística para avaliar a relevância prática das diferenças.

Em contextos de negócios, o *effect size* pode ser usado para tomar decisões informadas sobre a implementação de modelos de Machine Learning. Por exemplo, ao comparar dois modelos de classificação, um com uma acurácia ligeiramente maior e outro com custos de implementação significativamente maiores, o *effect size* pode ajudar a determinar se o ganho de acurácia é suficiente para justificar os custos adicionais. Em suma, o uso do *effect size* em Experimentação Contínua em Machine Learning ajuda a quantificar e interpretar a

magnitude das diferenças e relações entre variáveis, fornecendo uma base sólida para tomar decisões informadas e entender o impacto prático de resultados estatísticos.

Vejamos um exemplo em Python usando o *effect size* para comparar o desempenho de dois modelos de Machine Learning. Para isso, usaremos a métrica *Cohen's d* como medida de *effect size* . O valor de *Cohen's d* é calculado como a diferença entre as médias dos dois grupos, dividida pelo desvio padrão combinado, ou seja, indica quantos desvios padrão uma média está da outra.

Suponhamos que temos dois modelos A e B e queremos determinar se a diferença em suas acurácias é estatisticamente significativa e substancial:

```
import numpy as np
from scipy import stats

# Acurácias dos modelos A e B
modelo_A = np.array([0.78, 0.82, 0.85, 0.79, 0.81])
modelo_B = np.array([0.88, 0.92, 0.90, 0.89, 0.91])

# Calculando as médias e os desvios padrão das pontuações
media_A = np.mean(modelo_A)
media_B = np.mean(modelo_B)
desvio_padrao_A = np.std(modelo_A, ddof=1)  # Usando ddof=1 para calcular
o desvio padrão amostral
desvio_padrao_B = np.std(modelo_B, ddof=1)

# Calculando a diferença nas médias
diferenca_medias = media_B - media_A

# Calculando o Effect Size usando Cohen's d
effect_size = diferenca_medias / np.sqrt((desvio_padrao_A ** 2 +
desvio_padrao_B ** 2) / 2)

# Realizando um teste de hipótese para determinar a significância
```

```
estatística
t_stat, p_valor = stats.ttest_ind(modelo_A, modelo_B)

# Imprimindo os resultados
print(f'Média do Modelo A: {media_A:.2f}')
print(f'Média do Modelo B: {media_B:.2f}')
print(f'Diferença nas Médias: {diferenca_medias:.2f}')
print(f'Effect Size (Cohen\'s d): {effect_size:.2f}')
print(f'p-valor: {p_valor:.4f}')

# Interpretando o resultado
alpha = 0.05  # Nível de significância escolhido
if p_valor < alpha:
    print("Rejeita a hipótese nula: as acurácias são estatisticamente
diferentes")
else:
    print("Não há evidência suficiente para rejeitar a hipótese nula: as
acurácias são estatisticamente iguais")
```

A saída resultante será:

```
Média do Modelo A: 0.81
Média do Modelo B: 0.90
Diferença nas Médias: 0.09
Effect Size (Cohen's d): 4.02
p-valor: 0.0002
Rejeita a hipótese nula: as acurácias são estatisticamente diferentes
```

Valores de *Cohen's d* são geralmente interpretados como pequeno (0,2), médio (0,5) ou grande (0,8), embora essas categorias sejam orientações e o contexto específico deva sempre ser considerado. No caso do exemplo de código anterior, a média da acurácia de um modelo B está mais de 4 desvios padrão melhor que a do modelo A. Além da diferença estatisticamente significativa observada pelo p-valor do Teste t de Student, isso indica um efeito de grande magnitude.

De maneira geral, um *d* pequeno indica que a diferença entre os grupos, embora possivelmente significativa, é sutil; um *d* médio

sugere uma diferença mais notável e prática; enquanto um *d* grande aponta para uma diferença robusta e frequentemente de grande importância prática. Ao interpretar *Cohen's d*, é crucial avaliar não apenas a magnitude do efeito, mas também a relevância desse efeito no contexto da aplicação prática, garantindo decisões bem fundamentadas em Ciência de Dados e Machine Learning.

8.9 Considerações finais

Neste capítulo, exploramos os princípios fundamentais da Experimentação Contínua em Machine Learning. Discutimos conceitos como testes de hipótese, tipos de testes de hipótese, tipos de erros e a importância de controlar o nível de significância e o *effect size*. Abordamos também questões relacionadas com premissas associadas ao uso de alguns testes de hipótese, como a verificação de normalidade dos dados e a homogeneidade das variâncias. Além disso, fornecemos exemplos práticos de códigos para testes de hipótese paramétricos e não paramétricos, capacitando leitores e leitoras a aplicarem essas técnicas em cenários reais de Ciência de Dados.

Cabe ressaltar que a Experimentação Contínua no contexto de MLOps auxilia a melhoria e a adaptação contínua dos modelos de Machine Learning. MLOps envolve ainda um rigoroso monitoramento para avaliar a performance dos modelos e a qualidade dos dados, a utilização de ambientes segregados para testes e validação, além do versionamento de dados e modelos para facilitar a experimentação de forma segura. Neste contexto mais amplo de MLOps, a aplicação de estratégias de Engenharia de Software Contínua, como a Experimentação Contínua (Testes A/B) e lançamentos canários de versões beta com monitoração rigorosa permitem a implantação cuidadosa de novas versões de modelos, assegurando que permaneçam eficientes e alinhados às necessidades de negócio, que podem estar em constante evolução.

Ao chegarmos ao término deste livro de Introdução à Estatística para Ciência de Dados, esperamos que você agora esteja equipado com uma compreensão sólida dos conceitos fundamentais de Estatística Descritiva, Probabilidade e Inferência Estatística, e como esses conceitos se aplicam diretamente à Ciência de Dados. A jornada desde a introdução dos conceitos básicos até a aplicação de testes de hipóteses para a experimentação contínua em modelos de Machine Learning reflete não apenas a importância da Estatística como base da Ciência de Dados, mas também a evolução contínua do campo. Este livro visou ensinar os conceitos teóricos e a demonstrar como estes podem ser aplicados na prática. Encorajamos a continuidade do aprendizado e da exploração neste campo dinâmico, lembrando que a Estatística é uma ferramenta poderosa na tomada de decisões baseadas em dados.